Renesas Synergy™ Platform

GUIX™ Synergy Port Framework Module Guide

Introduction

This module guide will enable you to effectively use a module in your own design. Upon completion of this guide, you will be able to add this module to your own design, configure it correctly for the target application and write code, using the included application project code as a reference and efficient starting point. References to more detailed API descriptions and suggestions of other application projects that illustrate more advanced uses of the module are available on the Renesas Synergy Knowledge Base (as described in the References section at the end of this document), and should be valuable resources for creating more complex designs.

The Express Logic GUIX Synergy Port Module, sf_el_gx, is the Express Logic GUIX™ adaptation layer for Synergy MCU groups, which have graphics engines GLCDC, DRW (2DG engine), or a JPEG decode engine. The API supports graphics hardware engine setup for GUIX and supports graphics rendering and displaying accelerated by hardware engines. The module defines a full set of GUIX low-level display driver functions which draw graphics accelerated by DRW (2DG engine) or JPEG, or displays graphics with GLCDC (See the *GUIX User Guide*, Chapter 5: GUIX Display Drivers). The module encourages the hardware acceleration for graphics rendering but also allows software processing without hardware support.

Contents

1. GUIX™ Synergy Port Framework Module Features

The GUIX Synergy Port Framework module includes the following key functions:

- Adapts GUIX to the SSP Framework
- Attaches the SSP Display Interface driver to GUIX Display Driver Interface
- Allows GUIX to draw widgets accelerated by the Synergy D2W (2DG) engine
- Allows GUIX to draw widgets accelerated by the Synergy JPEG engine
- Supports double-buffer toggling control for screen transitions without tearing
- Supports screen rotation (90/180/270 degree)
- Supports various output color formats:
  - 32 bpp (ARGB8888, RGB-888)
  - 16 bpp (RGB565)
  - 8 bpp (8-bit palette (CLUT))
- Support for user callback functions

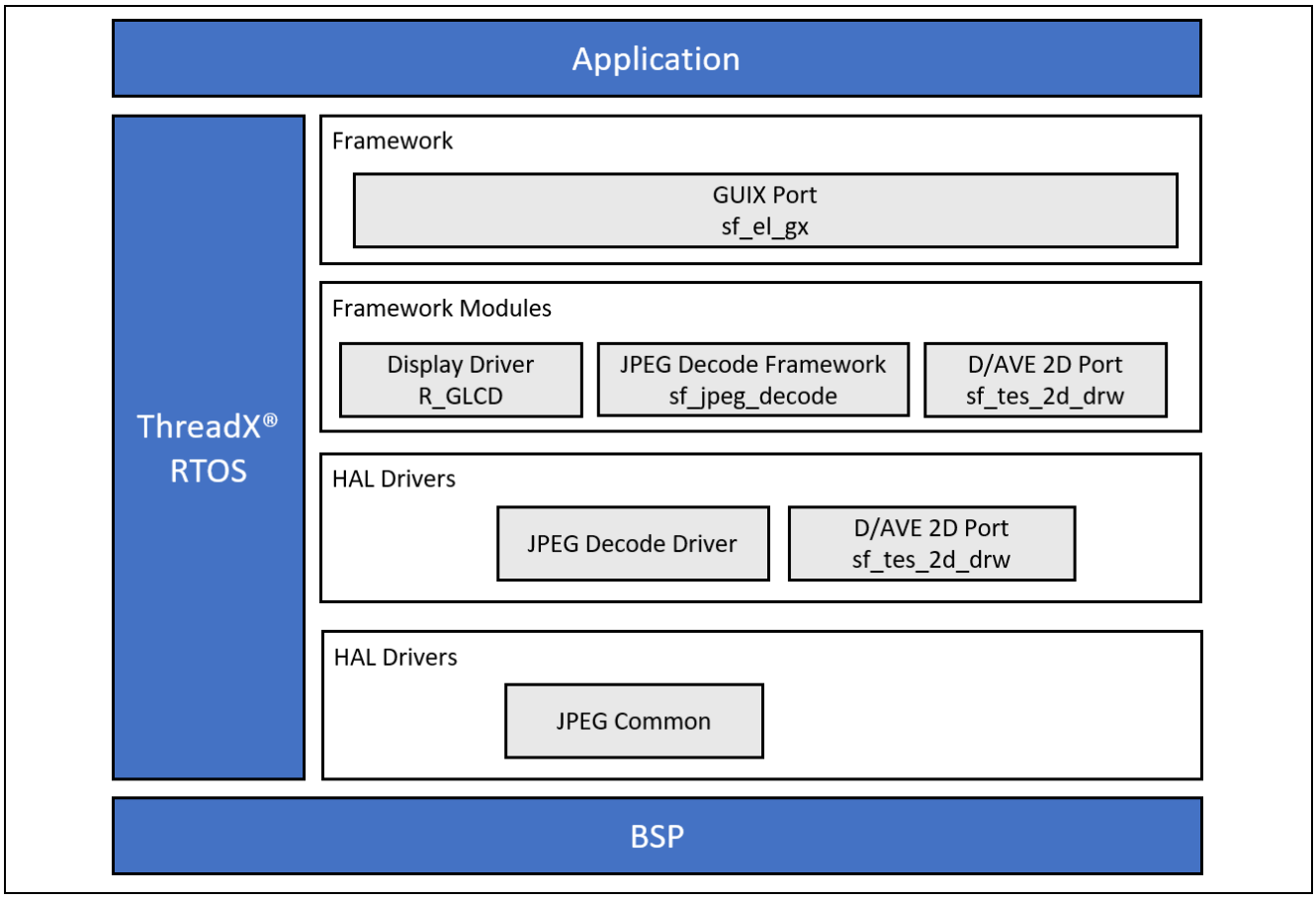

Figure 1. GUIX Synergy Port Framework Module Block Diagram

2. GUIX™ Synergy Port Framework Module APIs Overview

The GUIX Synergy Port Framework defines APIs for opening, closing, setup, and initialization. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Table 1. GUIX Synergy Port Framework Module API Summary

Function Name	Example API Call and Description
.open	`g_sf_el_gx.p_api->open(g_sf_el_gx.p_ctrl, g_sf_el_gx.p_cfg);` Opens the SF_EL_GX Module. The API can only be called from a thread. The API passes the configuration pointer to define low-level graphics device drivers, frame buffers, and register the user callback function. This function does not actually initialize low-level drivers. Instead, the API `setup` initializes the low-level drivers. The reason is in the explanation for `setup` below.
.close	`g_sf_el_gx.p_api->close(g_sf_el_gx.p_ctrl);` Closes the SF_EL_GX Module. This API closes the low-level drivers. Normally, the API is not called since GUIX will not be closed once initialized.
.versionGet	`g_sf_el_gx.p_api->versionGet(&version);` Returns the version of the Module in the version pointer.
.setup	`gx_studio_display_configure (MAIN_DISPLAY, g_sf_el_gx.p_api->setup, LANGUAGE_ENGLISH, MAIN_DISPLAY_THEME_1, &p_window_root);` This interface initializes low-level graphics device drivers. It must be passed to GUIX via the GUIX (Studio) service call `gx_studio_display_configure()` as a function pointer. GUIX then calls the API back. At that moment, the API configures the SSP device drivers based on the configuration passed by `open`. In this procedure to initialize low-level drivers, the API has the GUIX-compliant argument (GX_DISPLAY *) type and does not allow applying the detailed configuration of the SSP graphics device drivers generated from e^2 studio. The function `gx_studio_display_configure()` is located in a source file which is auto-generated by GUIX Studio.
.canvasInit	`g_sf_el_gx.p_api->canvasInit(g_sf_el_gx.p_ctrl, p_window_root);` This GUIX helper API determines the memory address of GUIX canvas. The API has an argument with type (GX_WINDOW_ROOT *). The API provides GUIX with the start address of canvas memory needed for the low-level graphics device drivers to draw/display images.

Note: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures and function variables, review the *SSP User's Manual* API References for the associated module.

Table 2. Status Return Values

Name	Description
SSP_SUCCESS	API call successful.
SSP_ERR_ASSERTION	NULL pointer error happens.
SSP_ERR_IN_USE	SF_EL_GX is in-use.
SSP_ERR_INTERNAL	Error happened in a Kernel service call.
SSP_ERR_NOT_OPEN	SF_EL_GX is not opened.
SSP_ERR_TIMEOUT	A task times out (or exceeds retry limit) before completion in display driver.
SSP_ERR_D2D_ERROR_DEINIT	Error occurred in D/AVE 2D driver.
GX_SUCCESS	Device driver setup is successfully done.
GX_FAILURE	Device driver setup failed.
SSP_ERR_INVALID_CALL	Function call was made when the driver is not in SF_EL_GX_CONFIGURED state.

Name	Description
SSP_ERR_D2D_RENDERING	The D/AVE 2D returns error at opening a display list buffer
SSP_ERR_INVALID_ARGUEMENT	Invalid non-pointer (e.g. parameter) input
SSP_ERR_UNSUPPORTED	Specified color format is not supported
SSP_ERR_D2D_ERROR_INIT	The D/AVE 2D returns error at the initialization.

Note: Lower-level drivers may return common error codes. Refer to the *SSP User's Manual* API References for the associated module for a definition of all relevant status return values.

3. GUIX™ Synergy Port Framework Module and JPEG Decode HAL Module Operational Overviews

The following figure shows components for a Synergy graphics solution and the flow of graphics data.

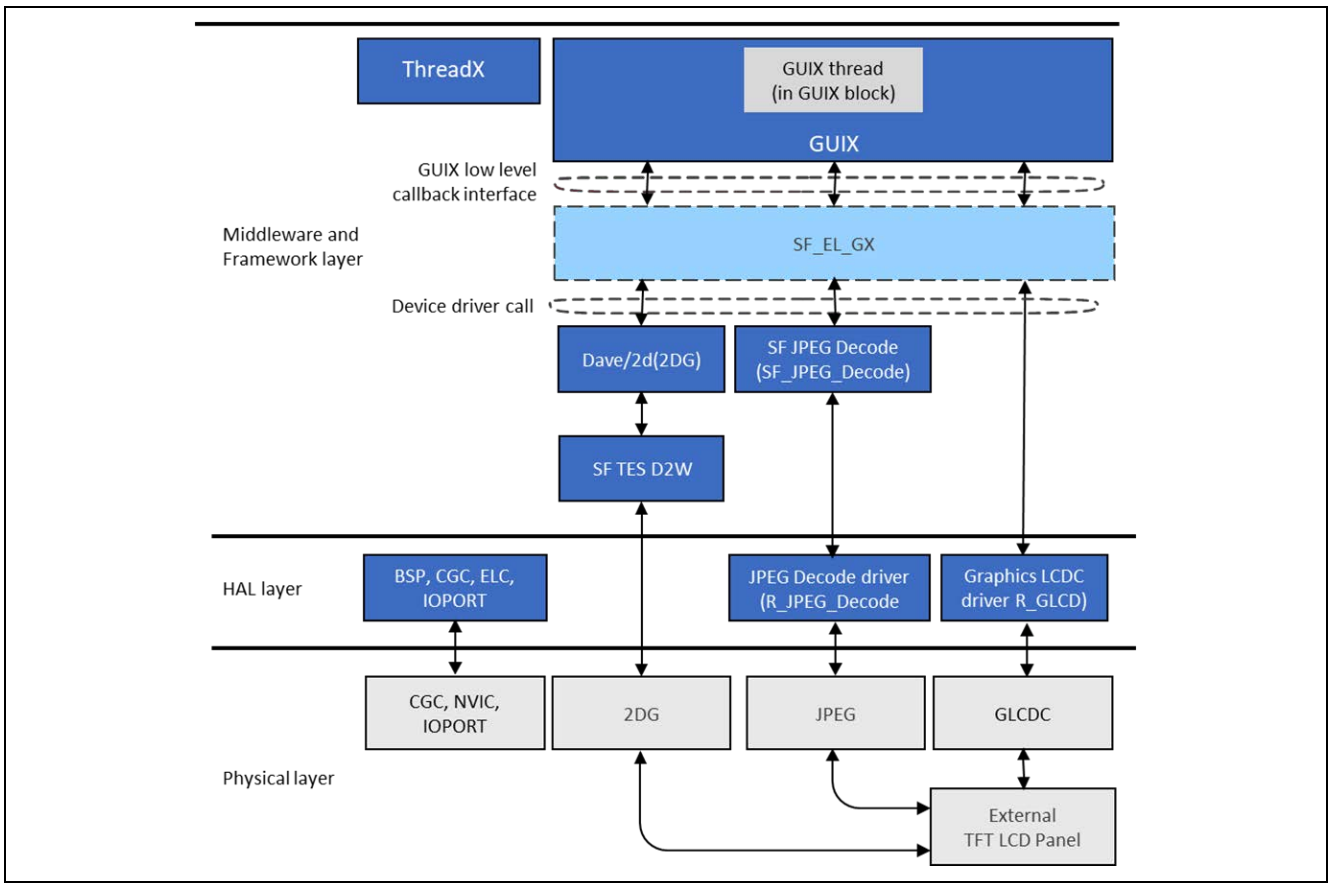

Figure 2. Graphics Solution Components

Module Initialization

The SF_EL_GX supports the Synergy graphics hardware setup, which is required to run the GUIX system. The module has a dependency on Express Logic GUIX™ and GUIX Studio™ generated code. The GUIX system initialization performs the following operations:

1. 'Open' SF_EL_GX module to initialize SF_EL_UX control block and pass module configurations.
2. Initialize GUIX Display object by calling GUIX Studio generated API `gx_studio_display_configure`. Through this API, the SF_EL_GX `setup` API is input to GUIX and Synergy graphics hardware setup will complete. Also, the root window initialized by GUIX is output to the user application.
3. Initialize the primary memory address of a GUIX Canvas Buffer by calling the `canvasInit` API.
4. Create the root window by calling the GUIX Studio generated API `gx_studio_named_widget_create`.
5. Show the root window by calling the GUIX API `gx_window_show`.
6. Start the GUIX system by calling the GUIX API `gx_system_start`.

Ping-Pong Frame Buffer Management

The SF_EL_GX module manages the buffer-toggling operation in a graphics system with a ping-pong frame buffer shown in the following figure. The graphics system is managed by the SF_EL_GX module. The module uses GUIX and the low-level display driver functions to draw an image (2D Drawing engine (DRW) or JPEG) and displays the image (DISPLAY module, for example, GLCDC). A design with a single frame buffer is also possible in the SF_EL_GX configuration. However, use two frame buffers to avoid tearing.

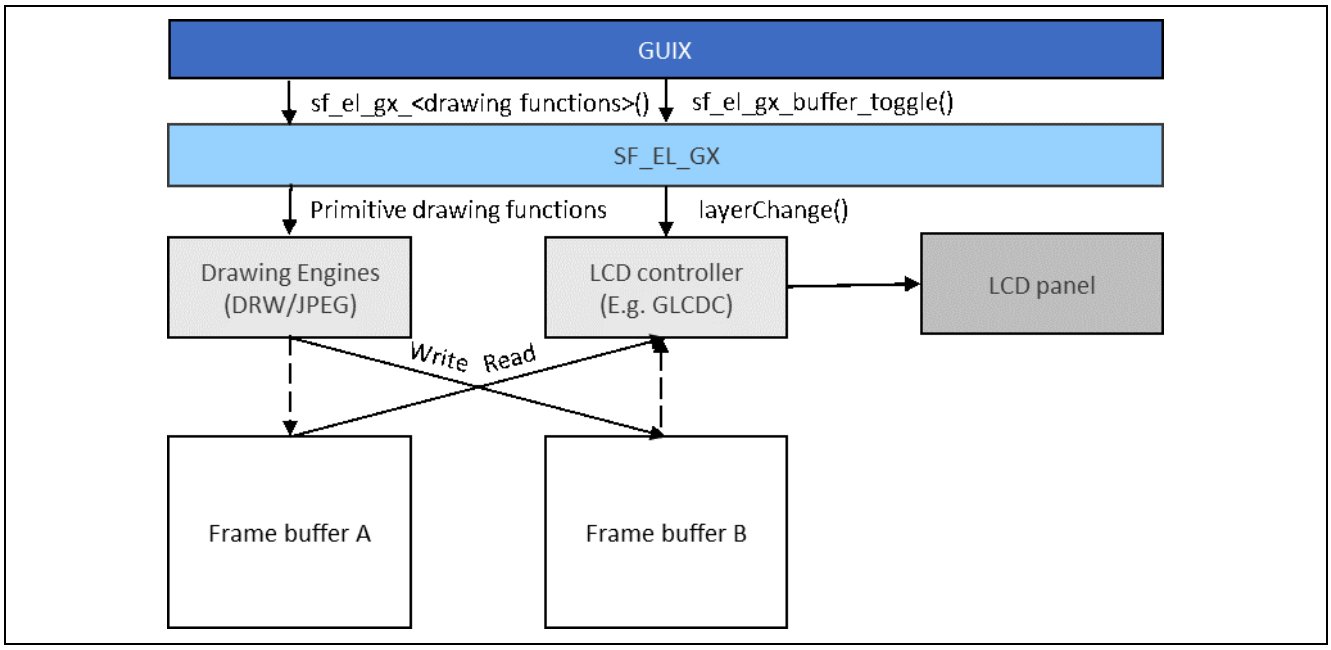

Figure 3. Ping-Pong Buffer

The diagram shows a case that does not use a GUIX Canvas buffer. When the Screen Rotation feature is enabled, GUIX draws screen updates to a GUIX Canvas buffer through the SF_EL_GX Module first, then the screen is to be copied to a non-visible side of the frame buffer. See Screen Rotation in the following section for details.

3.1 Important Operational Notes and Limitations for the GUIX™ Synergy Port Framework Module

3.1.1 GUIX Synergy Port Framework Module Operational Notes

Add the GUIX Source (`gx_src`) to your project if you want to change the default options of the GUIX library. To access these options, in the **Configurator Thread Stack** pane, click on **Add GUIX Source -> New -> GUIX Source**. The following GUIX Source options appear in the **Properties** window for the GUIX Source module:

- **Disable Multithread Support**: Disables the GUIX internal thread to take the GUIX system mutex for the GUIX system resources protection. If your system accesses GUIX only from a single thread, you can turn on this option to have GUIX omit the resource lock procedure and expect better system throughput. Unless you must tune your system performance, it is recommended to keep this option set to **No**.
- **Disable UTF8 Support**: Disables UTF8 format string encoding in GUIX and allows only 8-bit ASCII character plus Latin-1 code page character encoding.
- **System Timer (Milliseconds)**: Defines the GUIX timer rate. Allowed rates are integers larger than or equal to 10 ms.

Note: Check your SSP license to see whether you can view the GUIX Source files. For details about specific licenses, go to: synergygallery.renesas.com/user/license (login required).

GUIX Studio

When creating a project for Synergy, right-click the project and choose **Configure Displays**.

- To output resource and specification files directly to the Synergy project, browse to the `src` folder of your project for the Source, Header and Resource files.
- For **Target CPU**, choose **Renesas Synergy**.
- For **Toolchain**, choose **GNU**.
- Check the allocate canvas memory option.
- Set the **Display Number** to 1.
- Set the X and Y resolution for your hardware
- Set the GUIX library version to match the GUIX used in your Synergy project. For SSP 1.2.0 and 1.2.1, the current GUIX library is 5.3.2. Check the header information `gx_api.h` in your project for the version. The symbols defined in `gx_api.h`, `__GUIX_MAJOR_VERSION` and `__GUIX_MINOR_VERSION`, do not define the third digit.
- Set the main display name (default is **main_display**)

In the lower left of your GUIX Studio screen is the **Properties** view. This is where you set the properties for each widget, including the name of the widget (which is a window, button, scrollbar, text box, etc). It is also where you can set draw and event callback functions. These callback functions must be defined somewhere in your Synergy project source files or a compilation error will result. By default, these are set to NULL; if so, GUIX will use the default draw and event handling.

- Draw Function: If this field is blank, the standard drawing function for that widget type is used.
- Event Function: If blank, the standard event handling for this widget type is used.

Synergy 2D Drawing Engine Support

For older versions (before version 5.3.3), check **Enable Graphics Accelerator** in the **Synergy Advanced Settings** window.

If your project has selected a Pixelmap in the **Resource** pane option **Pixelmaps**, right-click on the image and choose **Edit Pixelmap** (or **Edit Settings** in version 5.3.3). For the **Output Format** in this window, select **Compress Output**. Do not select **Raw Format**. This configuration allows the GUIX Studio to generate Targa RLE format encoded image resource data. The D2W hardware can read this format and decode and draw the image on the frame buffer.

Synergy JPEG Support

In the **Configure Project** window, open the **Advanced Settings** window and select **Hardware JPEG Decoder** in the **Decoder Types JPEG** drop-down list.

Click the **Pixelmap** tab in the **Resource** pane. Right-click a previously added pixelmap to choose **Edit Pixelmap**. Select **Raw Format**. This configuration allows GUIX Studio to generate raw JPEG encoded image resource data. JPEG hardware can read this format, decode and draw the image on the frame buffer.

Adding Colors, fonts, pixelmaps and strings

In the right column of GUIX Studio, there are a series of drop-down lists for adding system and user defined strings, pixelmaps, colors, and fonts {optional]. For more details on how to manage these settings, please see the *GUIX Studio User Guide*. For the project in this module guide, there are no user defined options added except for strings.

Building/Rebuilding the Synergy Project

For any changes you make to GUIX Studio, you must generate the output files (Project -> Generate All Output Files). Then you must rebuild your synergy project for these changes to take effect.

SF_EL_GX Properties and the GUIX System

- GUIX Canvas Buffer

The GUIX Canvas Buffer is used to achieve the screen rotation of the screen image. The size of GUIX Canvas Buffer must be exactly the same as a frame buffer for the DISPLAY module.

- Screen Rotation

The GUIX Synergy Port Module is able to rotate the screen image drawn by GUIX. This feature is useful for a case such as a GUI screen design that should be a landscape shape, but the display panel hardware has

a portrait shaped screen. In this case, you can design your GUI in the landscape shape on GUIX Studio and display the screen on your display device with rotating by this module. The *Screen Rotation Angle* property may be 90, 180, 270 degrees, or counterclockwise. Dynamic screen rotation is not supported. To enable the screen rotation feature, the **GUIX Canvas Buffer** property must be set to **Used**. GUIX draws the screen update on a canvas first. The GUIX Port then processes the screen copy to a frame buffer, rotating the image in counterclockwise way. If the **Enable Synergy 2D Drawing Engine Support** property in the GUIX on the gx component is enabled, the rotation is processed by Synergy DRW with texture mapping. If not enabled, the rotation is processed by software copy.

The configuration: For **Screen Rotation angle** set to zero, **GUIX Canvas Buffer** can be set to **Used** but consumes extra bus bandwidth for a screen image copy. Therefore, the **GUIX Canvas Buffer** should be set to **Not used** for a **Screen Rotation angle** set to zero. GUIX then draws the screen update directly to a frame buffer so you can save memory and bus bandwidth.

- Size of JPEG Work Buffer

The JPEG work buffer trades off the JPEG decode speed against the buffer size. When a widget on the screen is formatted in JPEG, the JPEG work buffer is used as temporary storage memory to create the decoded image. If the buffer size is not large enough for decoding an entire image, JPEG decoding is performed in output buffer streaming mode. BitBLT (bit block transfer) by DRW decodes a piece of JPEG raster image in the buffer, then transfers it to the frame buffer.

Using a temporary work buffer is due to the JPEG hardware limitation for memory alignment; the number of pixels, and the requirement for alpha-blending if the widget has an alpha value. Since drawing an entire widget is completed with the repeated processing previously described, the data transfer overhead results in slower speed in decode processing, if the size of the JPEG work buffer is small. If the buffer size does not meet the minimum requirement, JPEG decoding will not be processed.

Stated another way, if you want to use the hardware decoder, more memory must be reserved into which the hardware decoder can output at least one MCU block of the decoded image. However, there are reasons for not using the hardware decoder. It has significant restrictions on JPEG image size, width modulus, format, output alignment, and so forth, and limits the processing of the JPEG image to decoding it into the frame buffer. The software decoder does not have those restrictions, but it takes up significantly more CPU time.

The minimum number in bytes for the **Size of JPEG Work Buffer** property must be: The number of pixels in the horizontal line x bpp (bytes per pixel) of the display format x 8 (lines).

For instance, if the decoded image is 800 pixels in a horizontal line and RGB565 format, the number is 800 x 2 x 8= 12,800 (bytes). To get better throughput, the **Size of the JPEG Work Buffer** property should be set to a value larger than the minimum value.

Event Notification and the GX_EVENT data type

GUIX events are requests made to one or more widgets to perform a particular action. For example, when a widget gains focus, the GX_EVENT_FOCUS_GAINED event is sent to the widget processing function.

When events are detected by the hardware, they get passed through to the GUIX event queue. At this point, the hardware data is transposed into a GX_EVENT data structure. The important fields of the GX_EVENT structure, which is defined in `gx_api.h`, are the `gx_event_type`, `gx_event_sender`, `gx_event_target`, and `gx_event_payload`.

- `gx_event_type` identifies the particular event class. The event type indicates if this is, for example, a GX_EVENT_PEN_DOWN event or a GX_EVENT_TIMER event.
- `gx_event_sender` contains the ID of the widget that generated the event if the event is a child-widget notification.
- `gx_event_target` is a pointer to a particular window or widget. If you want to send an event to a particular window, you should send a pointer to the widget.
- `gx_event_payload` is a union of various data types, and they are not all valid for every event type. You must examine the `gx_event_type` field first, to determine which of the payload data field is applicable. For GX_EVENT_PEN_DOWN and GX_EVENT_PEN_UP events, for example, the `gx_event_pointdata` field contains the x, y pixel coordinates the pen position. For timer events, the `gx_event_timer_id` contains the ID of the expired timer. Other payload data fields are utilized for other event types.

The following is a list of pre-defined GUIX event types:

- GX_EVENT_TERMINATE
- GX_EVENT_REDRAW
- GX_EVENT_SHOW
- GX_EVENT_HIDE
- GX_EVENT_RESIZE
- GX_EVENT_SLIDE
- GX_EVENT_FOCUS_GAINED
- GX_EVENT_FOCUS_LOST
- GX_EVENT_HORIZONTAL_SCROLL
- GX_EVENT_VERTICAL_SCROLL
- GX_EVENT_TIMER
- GX_EVENT_PEN_DOWN
- GX_EVENT_PEN_UP
- GX_EVENT_PEN_DRAG
- GX_EVENT_KEY_DOWN
- GX_EVENT_KEY_UP
- GX_EVENT_CLOSE
- GX_EVENT_DESTROY
- GX_EVENT_SLIDER_VALUE
- GX_EVENT_TOGGLE_ON
- GX_EVENT_TOGGLE_OFF
- GX_EVENT_RADIO_SELECT
- GX_EVENT_RADIO_DESELECT
- GX_EVENT_CLICKED
- GX_EVENT_LIST_SELECT
- GX_EVENT_VERTICAL_FLICK
- GX_EVENT_HORIZONTAL_FLICK
- GX_EVENT_MOVE
- GX_EVENT_PARENT_SIZED
- GX_EVENT_CLOSE_POPUP
- GX_EVENT_ZOOM_IN
- GX_EVENT_ZOOM_OUT
- GX_EVENT_LANGUAGE_CHANGE
- GX_EVENT_RESOURCE_CHANGE
- GX_EVENT_ANIMATION_COMPLETE
- GX_EVENT_SPRITE_COMPLETE
- GX_EVENT_TEXT_EDITED
- GX_EVENT_TGX_TIMER
- GX_EVENT_FOCUS_NEXT
- GX_EVENT_FOCUS_PREVIOUS
- GX_EVENT_FOCUS_GAIN_NOTIFY
- GX_EVENT_SELECT
- GX_EVENT_DESELECT
- GX_EVENT_PROGRESS_VALUE
- GX_EVENT_TOUCH_CALIBRATION_COMPLETE
- GX_EVENT_INPUT_RELEASE

The application can also add its own custom events, starting numerically after the constant GX_FIRST_APP_EVENT. Widgets with user-defined events must have a non-NULL event handler defined to process them.

At the hardware driver level, for example, where touchscreens and keypads detect PEN UP/DOWN and physical keystroke events, this data is ‘transposed’ at the driver level into a GX_EVENT data structure. This structure is where the event type and payload are stored to the event.

Then the driver calls the gx_system_event_send API where the event will be stored on the GUIX event queue. From here these events on the queue are routed by the GUIX system event dispatch mechanism for the GUIX widget handling. GUIX processes each event by either the default handler or, if defined, the custom event handler defined by the application. At the application level, the event handler receives as parameters a pointer to the widget instance and a pointer to the event structure itself.

The GX_SIGNAL macro is a macro that combines the ID of the widget that generated an event with the event type the widget has generated into a unique gx_event_type identifier code.

To catch signals generated by child widget, the parent event processing function will test for the event type using the GX_SIGNAL macro:

```
UINT my_window_event_process(GX_WINDOW *win, GX_EVENT *event)
{
  switch(event->gx_event_type)
  {
    case GX_SIGNAL(ID_CHECKBOX, GX_EVENT_TOGGLE_ON) :
       break;

    case GX_SIGNAL(ID_MY_SLIDER, GX_EVENT_SLIIDER_VALUE):
       break;

    default:
       gx_window_event_process(win, event);
  }
}
```

Where ID_MY_SLIDER and ID_MY_BUTTON are the widget IDs assigned to a widget in GUIX Studio, and GX_EVENT_CLICKED and GX_EVENT_SLIDER_VALUE are events generated by GUIX.

Note that widget IDs are optional in GUIX Studio. To use the GX_SIGNAL macro, you do need to supply a widget ID for the application to be able to determine what signal it has received. This is useful if the widget is a window, which contains several child widgets (for example, button or text box) and it needs to know which child widget has generated a particular event.

Example: a widget of type window with the event function callback window1_handler might be defined as follows in the application code:

```
UINT window1_handler(GX_WINDOW *widget, GX_EVENT *event_ptr)
{
  UINT result = gx_window_event_process(widget, event_ptr);
    switch (event_ptr->gx_event_type)
          {
          case GX_SIGNAL(ID_CHECKBOX, GX_EVENT_TOGGLE_ON) :

          /* Update parent window in which the widget belongs with new text.*/
          update_some_text(widget->gx_widget_parent, ID_WINDOWCHANGER,
             GX_STRING_ID_BUTTON_ENABLED);
             break;
```

Not all events can be generated by a widget. In fact, a widget is limited to certain events.

For example,

- GX_EVENT_PEN_XXXX events are generated by touch screen input drivers.
- GX_EVENT_KEY_xxx events are generated by keyboard drivers.

However, events such as GX_EVENT_SHOW and GX_EVENT_HIDE are generated internally when certain APIs cause widget status changes.

In Renesas Synergy™ platform, the application uses the Synergy SF Message framework to query a message queue used by the SF (software) Message framework for new events.

```
extern TX_QUEUE main_thread_message_queue;
extern const sf_message_instance_t g_sf_message0;
sf_message_header_t * p_message = NULL;
err = g_sf_message0.p_api->pend(g_sf_message0.p_ctrl,
&main_thread_message_queue, (sf_message_header_t **) &p_message,
TX_WAIT_FOREVER);
```

The pend function in the example above queries the hardware driver for new data. If there is a new 'message,' the message, which has an event structure containing event class and code for the particular hardware is returned to the application. (For example, for a touchscreen, the event class might be SF_MESSAGE_EVENT_CLASS_TOUCH and event code SF_MESSAGE_EVENT_NEW_DATA.)

The p_message contains an event class and code from the driver. The application can extract this from the p_message fields and determine which GUIX event it is, e.g. GX_EVENT_PEN_UP. Then, it can create the GX_EVENT from this and set the event type from the fields of the p_message instance.

```
gx_event->gx_event_type = GX_EVENT_PEN_UP;
gx_event->gx_event_sender = GX_ID_NONE;
gx_event->gx_event_target = 0;
gx_event->gx_event_display_handle = 0;
gx_event->gx_event_payload.gx_event_pointdata.gx_point_x = p_touch_payload->x;
gx_event->gx_event_payload.gx_event_pointdata.gx_point_y = (GX_VALUE)(320 –
   p_touch_payload->y);
```

Now, the application can forward the event to the GUIX message queue using the gx_system_event_send API. The project for this module guide does not use the message framework but the project does (www.renesas.com/us/en/software/D6003641.html).

Low level hardware drivers and the GX_EVENT data type

The driver hardware uses a ThreadX thread to periodically query the hardware for new data. When it detects new valid data, it verifies that it is valid data. It then translates that data into a type of user event filling and data, for example, cursor position associated with that event. Payload would be position coordinates in the example of a touchscreen. If this is a keypad device, the payload would store information for which keystrokes were detected.

SF_EL_GX Callback Function

The sf_el_gx instance has a user-defined callback function defined by the **Name of User Callback** property. By default, this is set to NULL. When the GUIX driver invokes this callback, which must be defined in the application code, the object pointer contains event, device and error information from the driver.

The device indicates type of hardware:

```
SF_EL_GX_DEVICE_NONE
SF_EL_GX_DEVICE_DISPLAY
SF_EL_GX_DEVICE_DRW
SF_EL_GX_DEVICE_JPEG
```

The event is one of the following:

```
SF_EL_GX_EVENT_ERROR
SF_EL_GX_EVENT_DISPLAY_VSYNC
SF_EL_GX_EVENT_UNDERFLOW
```

3.1.2 Synergy Port Framework Module Limitations

- SF_EL_GX is only applicable for the Synergy MCU with GLCDC (mandatory), the 2D Drawing engine, or the JPEG engine (optional)
- SF_EL_GX does not support a system with more than two frame buffers.
- SF_EL_GX supports only one GUIX canvas system.
- SF_EL_GX makes use of only one layer in DISPAY module.
- Do not access the TES D/AVE 2D module and TES D/AVE 2D Port module directly if GUIX uses the modules.
- Do not access the JPEG Decode Framework module and JPEG Decode HAL module directly if GUIX uses the modules.
- See the most recent SSP Release notes for additional limitations when using this module.

3.2 JPEG Decode HAL Module Operational Overview

The JPEG Decode HAL module is a high-level API for JPEG Decode processing implemented on r_jpeg. The module supports the Synergy JPEG Codec peripheral. It has the following features:

- Supports JPEG decompression.
- Supports polling mode that allows an application to wait for JPEG Decoder to complete.
- Supports interrupt mode with user-supplied callback functions.
- Configures parameters such as horizontal and vertical subsample values, horizontal stride, decoded pixel format, input and output data format, and color space.
- Obtains the size of the image prior to decoding it.
- Supports putting coded data in an input buffer and an output buffer to store the decoded image frame.
- Supports streaming coded data into JPEG Decoder module. This feature allows an application to read coded JPEG image from a file or from network without buffering the entire image.
- Configures the number of image lines to decode. This feature enables the application to process the decoded image on the fly without buffering the entire frame.
- Supports the input decoded format YCbCr444, YCbCr422, YCbCr420, YCbCr411.
- Supports the output format ARGB8888, RGB565.
- Returns error when the JPEG image's size, height and width don't meet the requirements.

The JPEG Decoder HAL module can be used in the Input Buffer Streaming mode or JPEG Output Buffer Streaming mode.

3.2.1 Input Buffer Streaming Mode Operational Description

In this scenario, the JPEG image data resides in a file, or is received from the network. The HAL-layer driver can handle this scenario without requiring the input data to be stored in memory first.

3.2.2 Output Buffer Streaming Mode Operational Description

In this scenario, the application needs to write the decoded image data to a file or to the network. The HAL-layer driver does not require the application to allocate memory for the entire frame. Instead the application may choose to decode one or more lines at a time. With this feature the amount of memory needed for the output data is greatly reduced.

3.2.3 JPEG Decode HAL Module Operational Notes

JPEG Decode Callbacks

A user callback function can be registered in the open API. If a user callback function is provided, the callback function will be called from the interrupt service routine (ISR) each time an interrupt happens. The argument of the callback function status can take the enumerated values listed below so that user can identify which event occurred in the decoding procedure.

Event Name	Event Condition
JPEG_DECODE_STATUS_ERROR JPEG	Decode module encountered an error.
JPEG_DECODE_STATUS_IMAGE_SIZE_READY JPEG	Decode obtained the image size of data to be decoded, and paused.
JPEG_DECODE_STATUS_INPUT_PAUSE JPEG	Decode paused waiting for more input data.
JPEG_DECODE_STATUS_OUTPUT_PAUSE JPEG	Decode paused after decoded the number of lines specified by user.
JPEG_DECODE_STATUS_DONE JPEG	Decode operation has successfully completed.

Note: Since a user callback function is called from an ISR, be careful not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

3.2.4 JPEG Decode HAL Module Limitations

- The JPEG Decode HAL module does not support JPEG Encode processing.
- Refer to the most recent *SSP Release Note* for the most up to date limitations for this module.

4. Including the GUIX™ Synergy Port Framework Module in an Application

This section describes how to include the GUIX Synergy Port Framework module in an application using the SSP configurator.

Note: It is assumed that you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the GUIX Synergy Port Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table..

Table 3. GUIX Synergy Port Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_el_gx0 GUIX Port on sf_el_gx	Threads	New Stack> Framework> Graphics> GUIX Port on sf_el_gx

When the GUIX on gx in added to the thread stacks as shown in the following figure, the configurator automatically adds any needed lower-level drivers. Any modules that need additional configuration information will be box text highlighted in red. Modules with a gray band are individual modules that stand alone. Modules with a blue band are shared or common and need only be added once, since they can be used by multiple stacks. Modules with a pink band can require the selection of lower level drivers. Sometimes these are optional or recommended and this is indicated in the block with the inclusion of this text. If the addition of lower level drivers is required, the module description will include **Add** in the text. Clicking on any pink banded modules will bring up the **New** icon and then will show the possible choices.

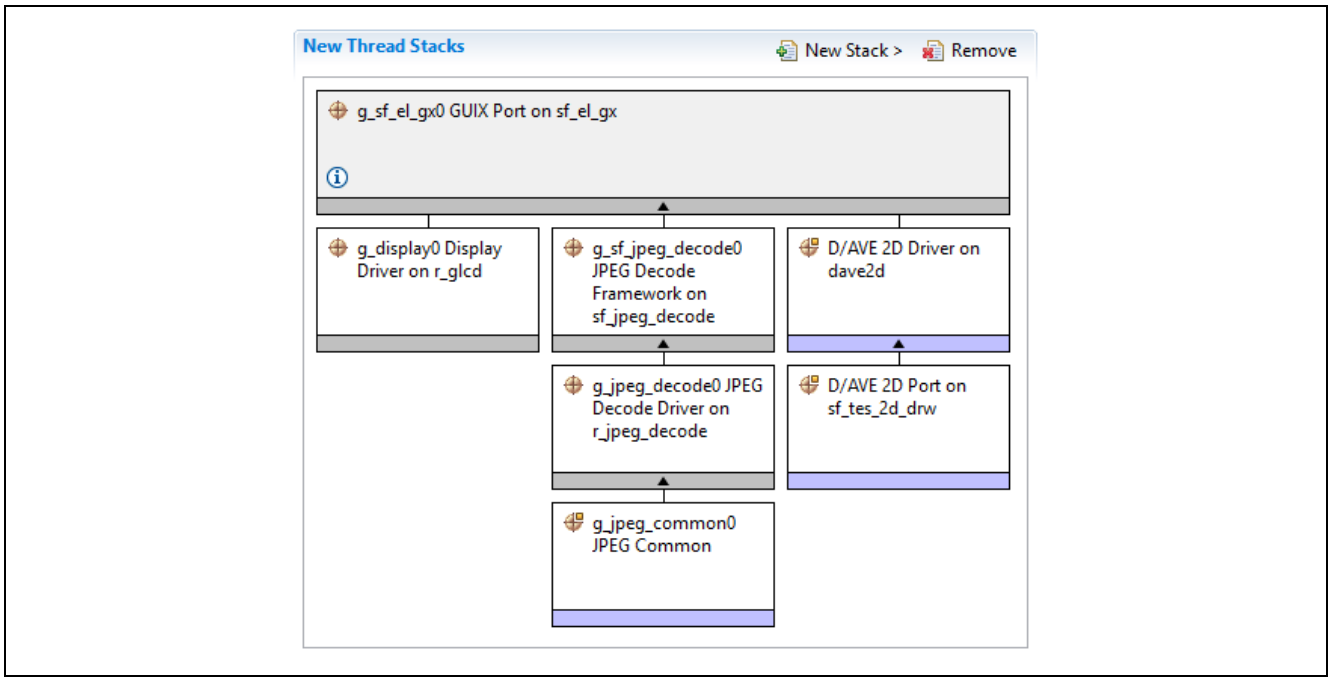

Figure 4. GUIX Synergy Port Framework Module Stack

5. Configuring the GUIX™ Synergy Port Framework Module

The GUIX Synergy Port Framework module must be configured by you for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and unavailable for changes, these are identified with a lock icon for the 'locked' property in the **Properties** window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority: this configuration setting is available within the **Properties** window of the associated module. Simply select the indicated module and then view the **Properties** window. The interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Also, note that the interrupt priorities listed in the **Properties** window in the ISDE will include an indication as to the validity of the setting based on the MCU targeted (CM4 or CM0+). The configuration properties in the following tables do not include this level of detail, but details are easily visible within the ISDE when configuring interrupt-priority levels.

Note: You may want to open your ISDE, create the module and explore the property settings in parallel with looking over the following configuration table settings. This helps to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Table 4. Configuration Settings for the GUIX Synergy Port Framework Module on GUIX on gx

ISDE Property	Value	Description
Enable Synergy 2D Drawing Engine Support	Yes, No Default: Yes	If Synergy 2D Drawing Engine (DRW) Support is enabled, the rotation is processed by Synergy DRW with texture mapping. If not enabled, the rotation is processed by software copy.
Enable Synergy JPEG Support	Yes, No Default: Yes	Enabling this support places restrictions on JPEG image size, width modulus, format, output alignment, and so forth. It is also limited to only decoding the JPEG image into the frame buffer. (The software decoder doesn't have any of those restrictions, but of course it uses more CPU time.)

Table 5. Configuration Settings for the GUIX Synergy Port Framework Module on sf_el_gx

Parameter	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_el_gx0	Name of SF_EL_GX instance which will be generated by ISDE. Specify the instance name of this module. Name must be a valid C symbol.
Name of Frame Buffer B (NULL allowed if consisting a single frame buffer system)	g_display0_fb_background[1]	Specify the name of another frame buffer. If you want to design your graphics system with a single frame buffer, set this parameter to NULL, or set same frame buffer name with parameter “Name of Frame Buffer A”. See Tearing in Single Buffer Designs
Name of User Callback function	NULL	Name of User Callback function invoked by the Module when events happen. It must be a valid C symbol and NULL is allowed.
Screen Rotation Angle (Clockwise)	0, 90, 180, 270 Default: 0	Angle of screen rotation (degree). If non-zero value is selected, screen rotation is enabled and GUIX draws screen image on a frame buffer, rotating the image with the angle in the counter clockwise way.
GUIX Canvas Buffer (required if rotation angle is not zero)	Not used; Used Default: Not used	If enabling the screen rotation, a canvas buffer must be used. The size of canvas buffer must be the same as a frame buffer for the display module.
Size of JPEG Work Buffer (valid if JPEG hardware acceleration enabled)	768000	The JPEG work buffer size in bytes. Value must be a valid integer value and zero is allowed to be set if JPEG acceleration is not used. Larger buffer size shortens the drawing time. See Size of JPEG Work Buffer
Memory section for GUIX Canvas Buffer	sdram, bss, … Default: sdram	Name of memory section where you want to allocate the GUIX Canvas Buffer. Enter a valid section name defined in the linker script file. Name must be a valid C symbol.
Memory section for JPEG Work Buffer	Sdram, bss, … Default: sdram	Name of memory section where you want to allocate the JPEG Work Buffer. Enter a valid section name defined in the linker script file. Name must be a valid C symbol.

Note**:** The example values and defaults are for a project using the Synergy S7G2. Other MCUs may have different default values and available configuration settings.

5.1 Configuration Settings for the GUIX™ Synergy Port Framework Module Low-Level Drivers

Typically, only a small number of settings must be modified from the default for lower level drivers as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and will be locked to prevent user modification. The following table identifies all the settings within the properties section for the module.

Table 6. Configuration for the GLCD HAL Module on r_glcd

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_display0	The name to be used for a GLCDC module control block instance. This name is also used as the prefix of the other variable instances.
Name of display callback function to be defined by user	NULL	Name must be a valid C symbol.
Input - Panel clock source select	Internal clock(GLCDCLK), External clock(LCD_EXTCLK) Default: Internal clock (GLCDCLK)	Choose the panel clock source depends on your system.
Input - Graphics screen1	Used, Not used Default: Used	Specify "Used" if the graphics screen N is used. Then the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is auto-generated by ISDE. If not using either of the graphics screens, specify "Not used". Then the frame buffer is not created. Note that there is no memory read access to the frame buffer when you specify "Not used", which reduces the consumption of bus bandwidth.
Input - Graphics screen1 frame buffer name	fb_background	Custom name for frame buffer.
Input - Number of Graphics screen1 frame buffer	2	Number of graphics selection.
Input - section where Graphics screen1 frame buffer allocated	sdram	Specify the section name to allocate the frame buffer. This is valid if "Input - Graphics screen1" is set as "Used."
Input - Graphics screen1 input horizontal size	800	Specify the number of horizontal pixels. Default value is the size for an image with 800x480 pixels
Input - Graphics screen1 vertical size	480	Specify the number of vertical pixels. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen1 input horizontal stride (not bytes but pixels)	800	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically, this parameter is set to same number as parameter 'input horizontal size'. Default value is the size for an image with 800x480 pixels.

ISDE Property	Value	Description
Input - Graphics screen1 input format	32bits ARGB888, 32bits RGB888, 16bits RGB565, 16bits ARGB1555, 16bits ARGB4444, CLUT 8, CLUT 4, CLUT 1 Default: 16bits RGB565	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.
Input - Graphics screen1 input line descending	On, Off Default: Off	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off".
Input - Graphics screen1 input line repeat	On, Off Default: Off	Specify "On" if expecting to repeatedly read a raster image which is smaller than the LCD panel size. Usually "Off". For details, see the description of Line Repeating function.
Input - Graphics screen1 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.
Input - Graphics screen1 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.
Input - Graphics screen1 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.
Input - Graphics screen1 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.
Input - Graphics screen1 layer background color Red	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer background color Green	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer background color Blue	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer fading control	None, Fade-in, Fade-out Default: None	Specify "Fade-In" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Fade-Out" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input - Graphics screen1 layer fade speed	0	Specify the number of frames for the fading transition to complete.

ISDE Property	Value	Description
Input - Graphics screen2	Used, Not used Default: Not used	Specify "Used" if the graphics screen N is used. Then the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is auto-generated by ISDE. If not using either of the graphics screens, specify "Not used". Then the frame buffer is not created. Note that there is no memory read access to the frame buffer when you specify "Not used", which reduces the consumption of bus bandwidth.
Input - Graphics screen2 frame buffer name	fb_foreground	Custom name for frame buffer.
Input - Number of Graphics screen2 frame buffer	2	Number of graphics selection.
Input - section where Graphics screen2 frame buffer allocated	sdram	Specify the section name to allocate the frame buffer. This is valid if "Input - Graphics screen1" is set as "Used."
Input - Graphics screen2 input horizontal size	800	Specify the number of horizontal pixels. Default value is the size for an image with 800x480 pixels
Input - Graphics screen2 vertical size	480	Specify the number of vertical pixels. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen2 input horizontal stride (not bytes but pixels)	800	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically, this parameter is set to same number as parameter 'input horizontal size'. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen2 input format	32bits ARGB888, 32bits RGB888, 16bits RGB565, 16bits ARGB1555, 16bits ARGB4444, CLUT 8, CLUT 4, CLUT 1 Default: 16bits RGB565	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.
Input - Graphics screen2 input line descending	On, Off Default: Off	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off".
Input - Graphics screen2 input line repeat	On, Off Default: Off	Specify "On" if expecting to repeatedly read a raster image which is smaller than the LCD panel size. Usually "Off". For details, see the description of Line Repeating function.
Input - Graphics screen2 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.
Input - Graphics screen2 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.

ISDE Property	Value	Description
Input - Graphics screen2 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.
Input - Graphics screen2 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.
Input - Graphics screen2 layer background color Red	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer background color Green	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer background color Blue	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer fading control	None, Fade-in, Fade-out Default: None	Specify "FadeIn" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Fade-Out" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input - Graphics screen2 layer fade speed	0	Specify the number of frames for the fading transition to complete.
Output - Horizontal total cycles	1024	Specify the total cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal active video cycles	800	Specify the number of active video cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal back porch cycles	46	Specify the number of back porch cycles in a horizontal line. Back porch starts from the beginning of Hsync cycles, which means back porch cycles contain Hsync cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal sync signal cycles	20	Specify the number of Hsync signal assertion cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches LCD panel on S7G2 PE-HMI1 board.

ISDE Property	Value	Description
Output - Horizontal sync signal polarity	Low active, High active Default: Low active	Select the polarity of Hsync signal to match your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical total lines	525	Specify number of total lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical active video lines	480	Specify the number of active video lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical back porch lines	23	Specify the number of back porch lines in a frame. Back porch starts from the beginning of Vsync lines, which means back porch lines contain Vsync lines. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical sync signal lines	10	Specify the Vsync signal assertion lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical sync signal polarity	Low active, High active Default: Low active	Select the polarity of Vsync signal to match to your system. Default setting matches LCD panel on S7G2 PE-HMI1 board.
Output - Format	24bits RGB888, 18bits RGB666, 16bits RGB565, 8bits serial Default: 24bits RGB888	Specify the graphics screen output format to match to your LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Endian	Little endian, Big endian Default: Little endian	Select data endian for output signal to LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Color order	RGB, BGR Default: RGB	Select data order for output signal to LCD panel. The order of blue and red can be swapped if needed. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Data Enable Signal Polarity	Low active, High active Default: High active	Select the polarity of Data Enable signal to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Sync edge	Rising Edge, Falling Edge Default: Rising Edge	Select the polarity of Sync signals to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Background color alpha channel	255	Specify the background color of the background screens.
Output - Background color R channel	0	Specify the background color of the background screens.

ISDE Property	Value	Description
Output - Background color G channel	0	Specify the background color of the background screens.
Output - Background color B channel	0	Specify the background color of the background screens.
CLUT	Used, Not used Default: Not used	Specify "Used" if selecting CLUT formats for a graphics screen input format. Then, a buffer named "CLUT_buffer" for the CLUT source data is generated in the ISDE auto-generated source file.
CLUT - CLUT buffer size	256	Specify the number of entries for the CLUT source data buffer. Each entry consumes 4 bytes (1 word). Words of CLUT source data specified by this parameter are generated in the ISDE auto-generated source file.
TCON - Hsync pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON0	Select the TCON pin used for the Hsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - Vsync pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON1	Select TCON pin used for Vsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - DataEnable pin select	Not used, LCD_TCON0, LCD_TCON1, LCD_TCON2, LCD_TCON3 Default: LCD_TCON2	Select TCON pin used for DataEnable signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - Panel clock division ratio	1/1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/12, 1/16, 1/24, 1/32 Default: 1/8	Select the clock source divider value. See the note at bottom of this table about the source clock for the pixel clock.
Color correction - Brightness	Off, On Default: Off	Specify "On" when performing brightness control. If specifying "Off", the setting below does not affect the output color.
Color correction - Brightness R channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Brightness G channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Brightness B channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Contrast	Off, On Default: Off	Specify "On" when performing contrast control. If specifying "Off", the setting below does not affect the output color.

ISDE Property	Value	Description
Color correction - Contrast(gain) R channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Contrast(gain) G channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Contrast(gain) B channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Gamma correction(Red)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain R[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold R[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma correction(Green)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the green channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain G[0-15]	0	Set the gain value for the green channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold G[0-15]	0	Set the threshold value for the green channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).

ISDE Property	Value	Description
Color correction - Gamma correction(Blue)	Off, On Default: Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the blue channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain B[0-15]	0	Set the gain value for the blue channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold B[0-15]	0	Set the threshold value for the blue channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Dithering	Off, On Default: Off	Dithering enable. Specify "On" when applying the dither effect to reduce the banding in case of selecting RGB666 or RGB565 output formats. Dithering can be applied when converting. If specified "Off", the settings for dithering below do not affect the output. For details on the dither effect, see Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Mode	Truncate, Round off, 2x2 Pattern Default: Truncate	Specify the dither mode. For detail, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern A	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For more details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern B	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For more details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHB) in the hardware manual.
Dithering - Pattern C	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For more details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHC) in the hardware manual.
Dithering - Pattern D	Pattern 00, Pattern 01, Pattern 10, Pattern 11 Default: Pattern 11	Specify the dither pattern for 2X2 pattern mode. For more details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHD) in the hardware manual.

ISDE Property	Value	Description
Misc - Correction Process Order	Brightness and Contrast then Gamma, Gamma then Brightness and Contrast Default: Brightness and Contrast then Gamma	Specify the color correction processing order if needed.
Line Detect Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	The driver needs valid interrupt priority setting and it won't work if disabled.
Underflow 1 Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	The driver needs valid interrupt priority setting and it won't work if disabled.
Underflow 2 Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	The driver needs valid interrupt priority setting and it won't work if disabled.

Table 7. Configuration for the JPEG Decode Framework Module on sf_jpeg_decode

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter checking.
Name	g_sf_jpeg_decode0	The name to be used for a JPEG Decode Framework module instance.

Table 8. Configuration for the JPEG Decode HAL Module on r_jpeg

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_jpeg_decode0	The name to be used for a JPEG Decode module instance.

ISDE Property	Value	Description
Byte Order for Input Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word-Byte Swap (8)(7)(6)(5)(4)(3)(2)(1) Default: Normal Byte order	Specify the byte order for input data. The order is swapped as specified in every 8-byte.
Byte Order for Output Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8), Byte Swap (2)(1)(4)(3)(6)(5)(8)(7), Word Swap (3)(4)(1)(2)(7)(8)(5)(6), Word-Byte Swap (4)(3)(2)(1)(8)(7)(6)(5), Longword Swap (5)(6)(7)(8)(1)(2)(3)(4), Longword-Byte Swap (6)(5)(8)(7)(2)(1)(4)(3), Longword-Word Swap (7)(8)(5)(6)(3)(4)(1)(2), Longword-Word-Byte Swap (8)(7)(6)(5)(4)(3)(2)(1) Default: Normal Byte order	Specify the byte order for output data. The order is swapped as specified in every 8-byte.
Output Data Color Format	Pixel Data RGB565 format, Pixel Data ARGBB888 format Default: Pixel Data RGB565 format	Specify the output data format.
Alpha value to be applied to decoded pixel data (only valid for ARGB8888 format)	255	Specify the alpha value for the output data format (only valid for ARGB8888 format).
Name of user callback function	NULL	Specify the name of user callback function.
Decompression Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Decompression interrupt priority selection.
Data Transfer Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	Data transfer interrupt priority selection.

Table 9. Configuration for the D/AVE 2D Driver on dave2d

ISDE Property	Value	Description
No configurable settings		

Table 10. Configuration for the D/AVE 2D Port on sf_tes_2d_drw

ISDE Property	Value	Description
Work memory size for display lists in bytes	32768	Work memory size for display lists selection
DRW Interrupt Priority	Priority 0 (highest), Priority 1:2, Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX), Priority 4:14 (CM4: valid, CM0+: invalid), Priority 15 (CM4 lowest - not valid if using ThreadX, CM0+: invalid) Default: Disabled	DRW INT selection

Note: The example values and defaults are for a project using the Synergy S7G2 Family. Other MCUs may have different default values and available configuration settings.

5.2 GUIX™ Synergy Port Framework Module Clock Configuration

The GUIX Synergy Port Module is a logical module and therefore does not require any hardware setting except setting the ARM Cortex®-M core SysTick timer.

5.3 GUIX™ Synergy Port Framework Module Pin Configuration

The GUIX Synergy Port Module is a logical module and therefore does not require pin settings.

6. Using the NetX™ or GUIX™ Synergy Port Framework Module in an Application

The following important settings are made in the Synergy Configurator and are used to initialize the module:

- Setup GLCDC configurations including the module clock setting and GLCDC interrupt priority. Typically, the configuration can be auto-generated through Synergy Configurator.
- Setup 2D Drawing engine or JPEG engine configurations including the module clock setting and hardware interrupt priorities. Typically, the configuration can be auto-generated through Synergy Configurator.

The typical steps in using the GUIX Framework module in an application are:

1. Initialize the SF_EL_GX control block and pass module configuration settings by calling the `open` API.
2. Complete initialization by calling the GUIX Studio generated `gx_studio_display_configure` API and pass the SF_EL_GX setup function shown as follows. This function call completes the initialization of Synergy graphics hardware accelerators. Obtain the address of the root window initialized by GUIX through the call.

```
gx_studio_display_configure (MAIN_DISPLAY,
                             g_sf_el_gx0.p_api->setup,
                             LANGUAGE_ENGLISH,
                             MAIN_DISPLAY_THEME,
                             &p_window_root);
```

3. Initialize the primary memory address GUIX Canvas buffer by calling the `canvasInit` API.
4. Create the root window by calling the GUIX Studio generated `gx_studio_named_widget_create` API.
5. Show the root screen by calling the GUIX `gx_widget_show` API.
6. Start the GUIX system by calling the GUIX `gx_system_start` API.

Once GUIX system is started, the SF_EL_GX module is driven under GUIX control. The application need not execute any operations after this.

These common steps are illustrated in a typical operational flow diagram in the following figure:

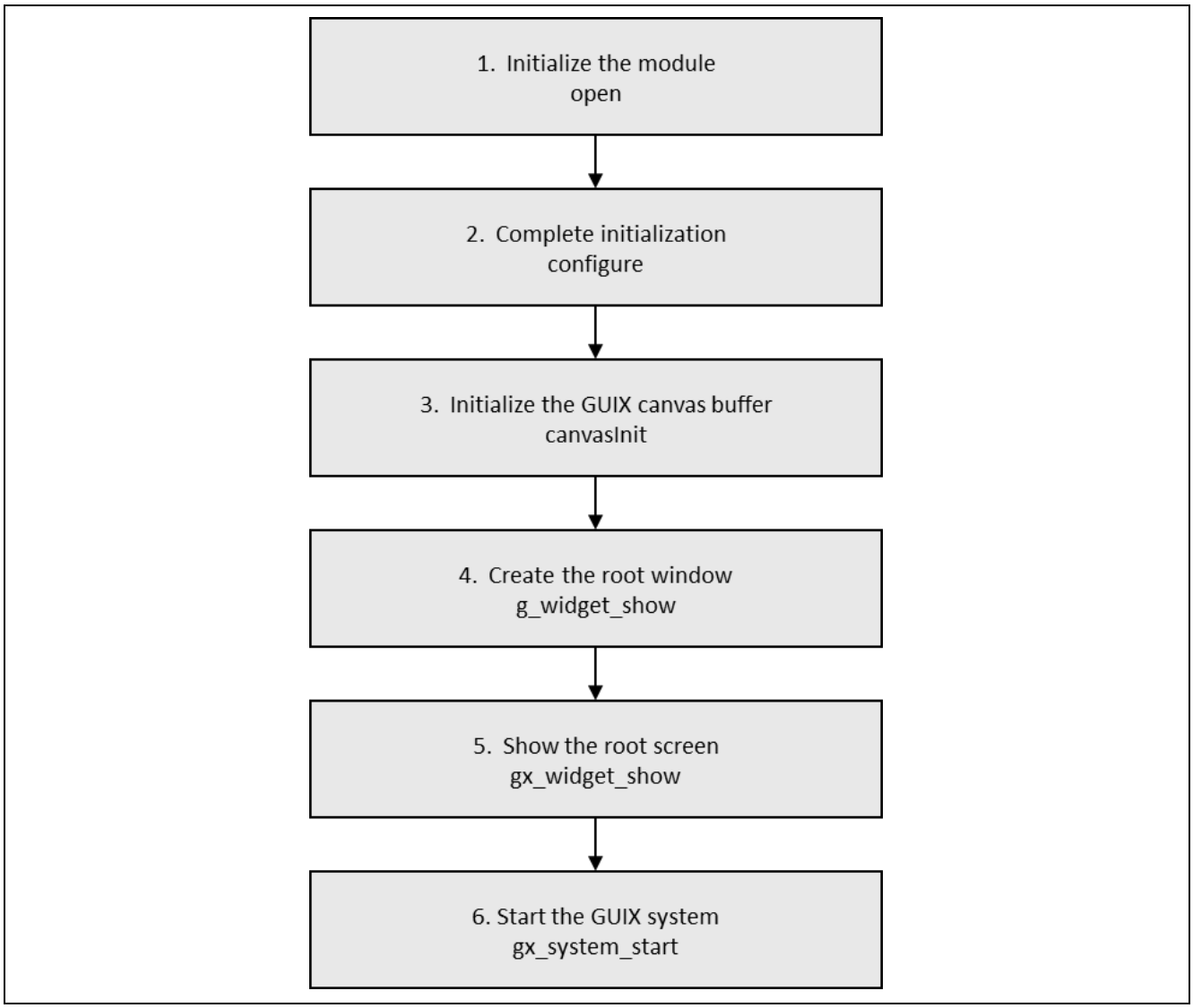

Figure 5. Flow Diagram of a Typical GUIX Synergy Port Framework Module Application

Once the JPEG Decode HAL module has been configured and the files generated, the JPEG Decode is ready to be used in an application. The typical steps in using the JPEG Decoder HAL module in an application are initializing the JPEG Decode using the `open` API, configure the horizontal stride, image sub-sample, input buffer and output buffer, once the input and output buffers are set, JPEG codec trigger the decode operation and store the decoded image to the output buffer, the `statusGet` API can be used to poll the status of JPEG operation.

The typical steps in using the JPEG Decode HAL module in an application are:

1. Initialize the JPEG Decode HAL module using the `open` API.
2. Set the horizontal stride using the `horizontalStrideSet` API.
3. Set vertical and horizontal image sub-sample using the `imageSubsampleSet` API.
4. Set the input buffer address (which contains the JPEG image) using the `inputBufferSet` API.
5. Set the output buffer (should be large enough to hold the raw image data) using the `outputBufferSet` API.
6. The `statusGet` API can be used to get the JPEG operation, `statusGet` API return an enumerated value (described above) to notify the user. Status JPEG_DECODE_STATUS_DONE from `statusGet` API shows that the decode operation is complete.
7. Operate on the received raw image data as needed by the application.

These common steps are illustrated in a typical operational flow diagram in the following figure:

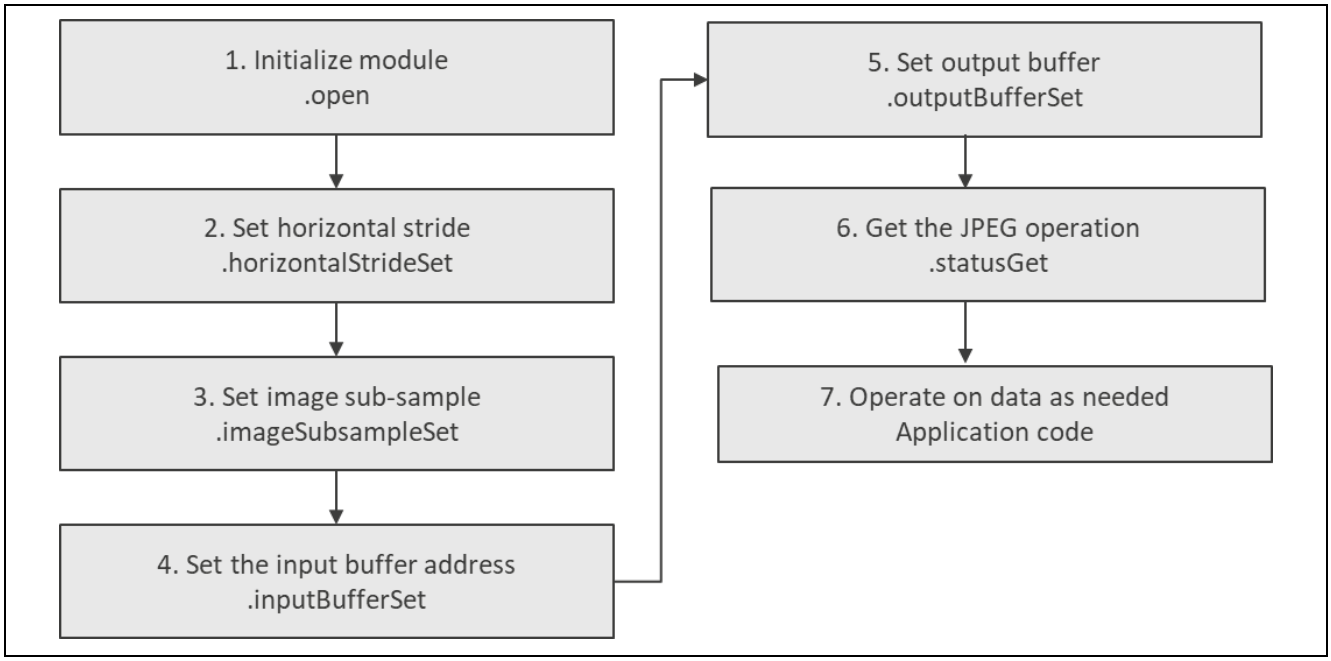

Figure 6. Flow Diagram of Typical Operation

7. GUIX™ Synergy Port Framework Module Application Project

The application project associated with this module guide demonstrates the steps in a full design. The project can be found using the link provided in the References section at the end of this document. You may want to import and open the application project within the ISDE and view the configuration settings for the GUIX Synergy Port Framework module. You can also read over the code (in `my_gui_thread_entry.c`) which is used to illustrate the GUIX Synergy Port Framework module APIs in a complete design.

The application project main thread entry initializes the GUIX system and low-level drivers for the display and LCD hardware. Then it runs one of two different tests depending on whether it is set to LCD_TEST or GUIX_TEST mode. If the mode is set to GUIX_TEST, it uses the resource and specification files generated by GUIX Studio. If it is set to LCD_TEST, it creates a simple three color bar image and calls the LCD services in the `lcd_setup.c` file to draw the image on the LCD screen. The application keeps an error counter for any API that returns a non-success status return or a GUIX callback that indicates an error. If SEMI_HOSTING is defined, the debug `printf` output indicates a successful or failed test result.

Table 11. Software and Hardware Resources Used by the Application Project

Resource	Revision	Description
e^2 studio	7.5.1 or later	Integrated Solution Development Environment (ISDE)
SSP	1.7.0 or later	Synergy Software Platform
IAR EW for Renesas Synergy	8.23.3 or later	IAR Embedded Workbench for Renesas Synergy
SSC	7.5.1 or later	Synergy Standalone Configurator
SK-S7G2	V3.0 to v3.3	Starter Kit
GUIX Studio	5.4.0 or later	Stand-alone Windows tool to create bitmaps

A simple flow diagram of the application project is given in the following figure:

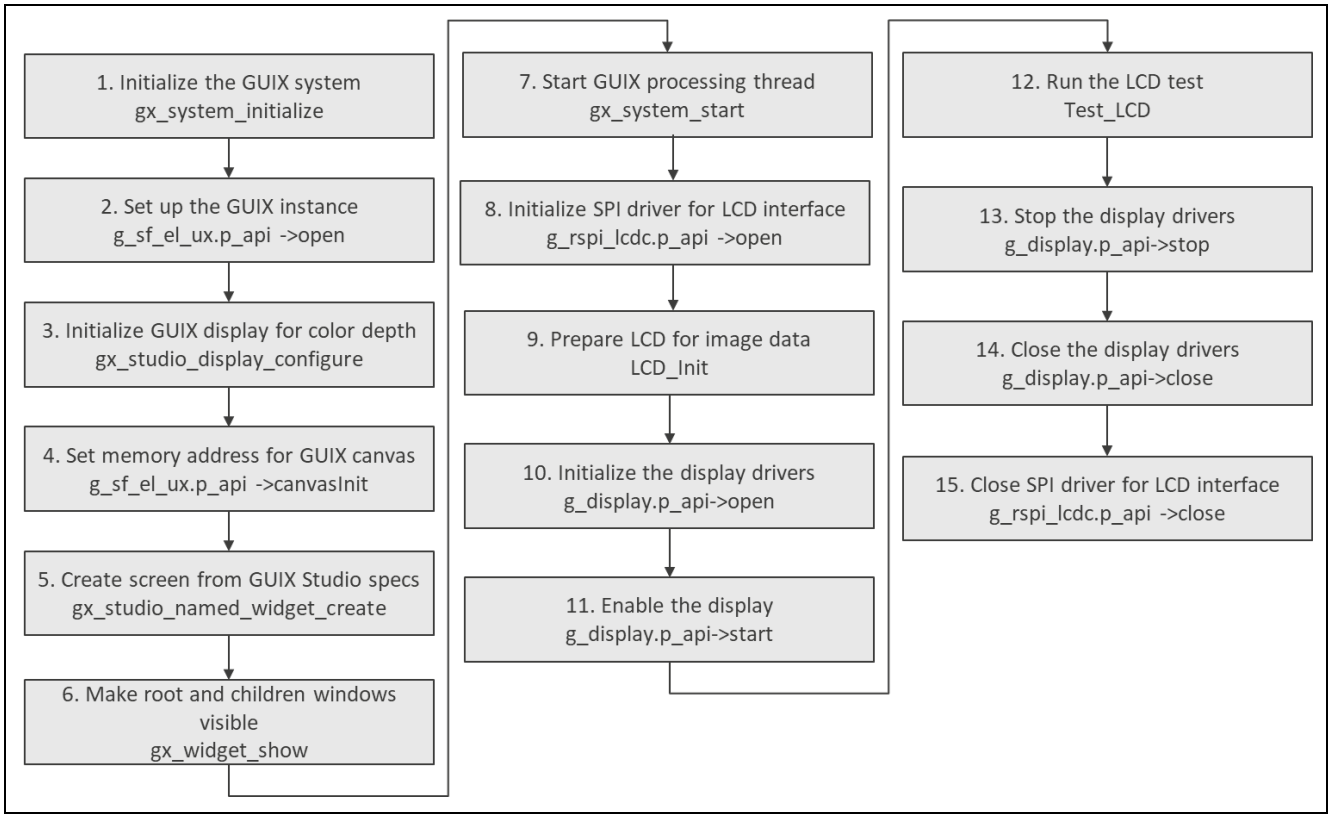

Figure 7. GUIX Synergy Port Framework Module Application Project Flow Diagram in LCD_TEST Mode

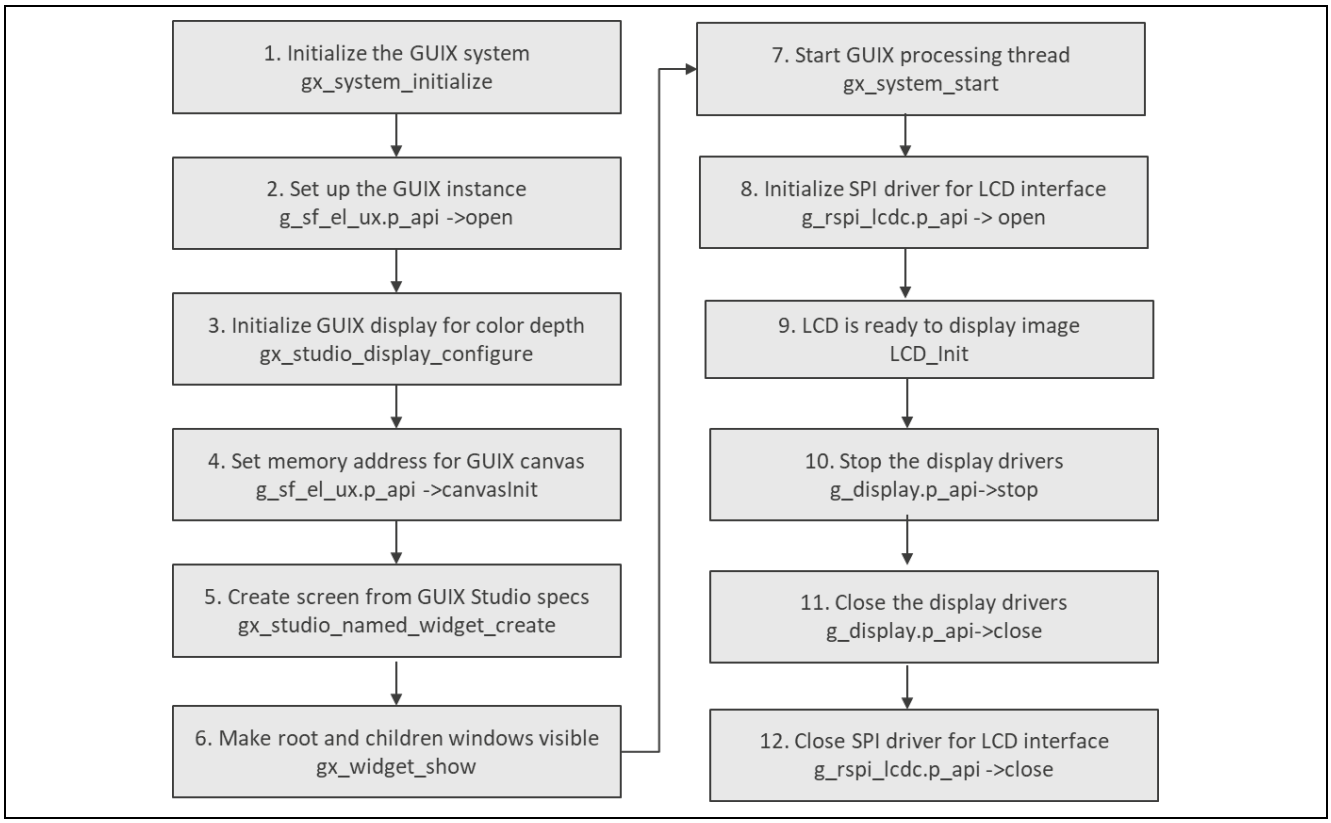

Figure 8. GUIX Synergy Port Framework Module Application Project Flow Diagram in GUIX_TEST Mode

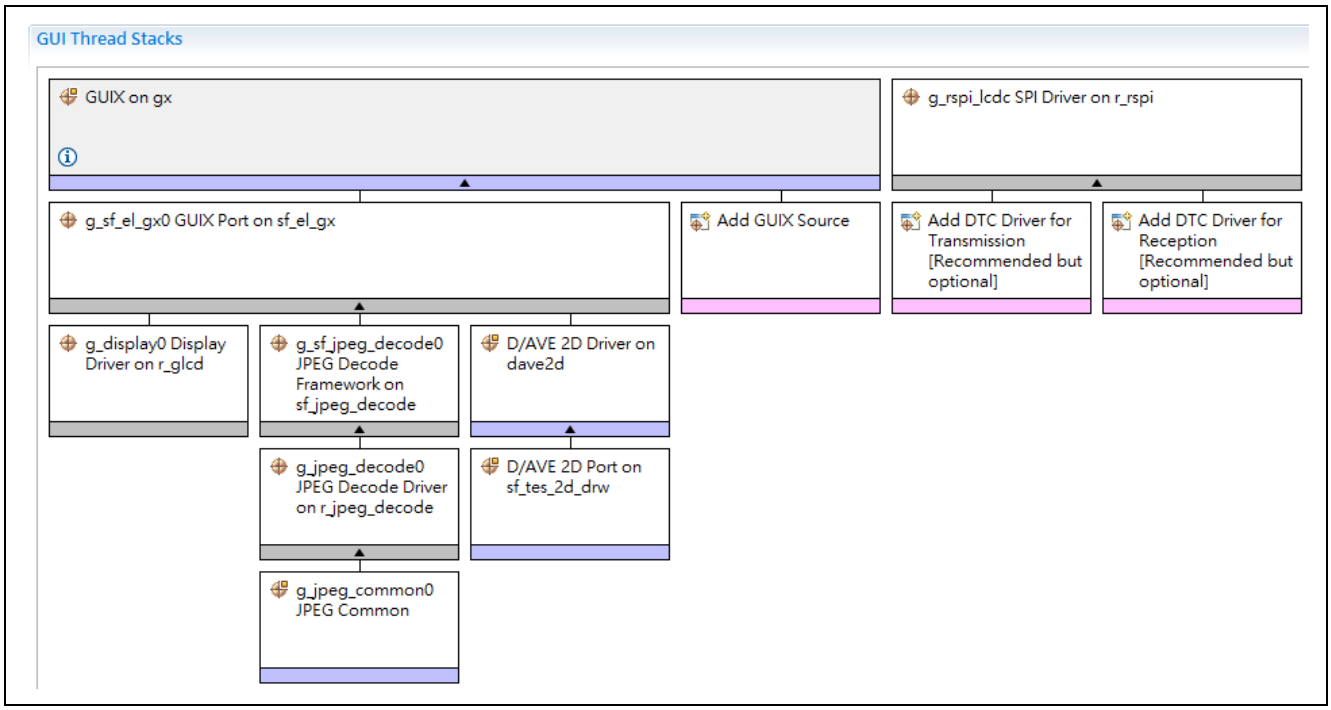

Figure 9. GUIX Synergy Port Framework Module Application Project Components with Interface to LCD Hardware

The r_rspi_lcdc instance of the SPI driver is defined to have a callback, `my_lcd_spi_callback`. This function is defined in `guix_driver_sf_el_gx_mg_ap.c` and when it is called it releases the semaphore to allow the LCD to continue processing. This callback allows the SPI to return information about the event transfer to the application in the spi_callback_args_t pointer input. `my_lcd_spi_callback` uses the semaphore created in the e^2 studio **Thread Stack** pane to control reads and writes to the SPI driver instance.

The `my_guix_thread_entry.c`, `guix_driver_sf_el_gx_mg_ap.c` , `guix_driver_sf_el_gx_mg_ap.h`, `my_gui_event_handler.c`, `lcd.h`, and `lcd_setup.c` files are located in the project once it has been imported into the ISDE. You can open these files within the ISDE and follow along with the description provided to help identify key uses of APIs.

The `my_guix_thread_entry.c` file has the thread entry function. It initializes the printf debug output. Then it makes two calls to the `guix_driver_test_image_draw` function defined in `guix_driver_sf_el_gx_mg_ap.c`, one with the mode set to LCD_TEST and the other with the mode set to GUIX_TEST. Then it prints the results of those tests for whether any errors occurred and the number of display events that the GUIX notified the application.

GUIX_TEST

In this test mode, the resource files, (which may contain pixelmap data) and specification files (which define initial screen state) are created by GUIX Studio, and saved to the project `src` folder. In this project, there are four GUIX Studio files that need to be included in the project `src` folder:

- `guix_for_mg_ap_resources.c`
- `guix_for_mg_ap_resources.h`
- `guix_for_mg_ap_specifications.c`
- `guix_for_mg_ap_specifications.h`

The file names are based on the name of the project in GUIX Studio, which for this project was guix_for_mg_ap.

The `guix_driver_sf_el_gx_mg_ap.c` #includes header files for GUIX Studio generated files, `gx_api.h` for GUIX services, `lcd.h` for LCD image processing, and `my_gui_thread.h` which is the Synergy generated header file based on the Thread Stack components. It also defines several function prototypes, including a callback for the GUIX system, `my_guix_callback`. This callback enables GUIX to notify the application of events and errors while running GUIX. This property for setting this callback is the **Name of User Callback** property in the `g_sf_el_gx0` component of the project.

guix_driver_sf_el_gx_mg_ap.c defines a pointer to a root window, p_window_root, which will be used in GUIX and GUIX Studio API. For each visible canvas created, a root window must be created for that canvas. This special window basically acts as a container for all the top-level application windows and widgets. See the *GUIX User Guide* listed in the Reference section for more details.

guix_driver_test_image_draw is the function that performs the processing for the image for the LCD panel:

1. It begins by initializing the GUIX system by calling the gx_system_initialize API.
2. The GUIX instance g_sf_el_gx0 module gets set up (creates mutex to lock the GUIX driver during a context update, creates a semaphore for rendering and displaying synchronization, checks the GUIX canvas memory, screen rotation and other details, and initializes the g_sf_el_gx0 control block):
   A. `g_sf_el_gx0.p_api->open()`
3. Next, the gx_studio_display_configure API initializes the display driver. It puts the components of the image together in the display object, including the theme, color palette, font, and language.
   This API installs the requested GUIX resource theme, which specifies the fonts, colors, and pixelmap data associated with each Resource ID. The API further installs the application string table, and defines the currently active language. The parameter MAIN_DISPLAY identifies which display is being configured (in case the application defines multiple hardware displays). The parameter MAIN_DISPLAY_THEME_1 defines the resource theme to be installed, since applications can define multiple resource themes. Finally, the parameter LANGUAGE_ENGLISH defines the language that is initially active, since GUIX applications can define multiple languages and select the active language at runtime.

   These resources are defined in GUIX Studio project files listed above.
4. The canvasInit API sets the memory address for the GUIX canvas, which is the buffer area into which the GUIX display driver will draw.
5. The gx_studio_named_widget_create API creates the “splash_screen” widget using the data supplied in the GUIX Studio generated specifications file. This will be the primary screen.
6. The gx_widget_show API makes the root window, and all children of the root window, visible.
7. Now the application calls the gx_system_start API which starts the GUIX system thread. This thread is responsible for dispatching events posted to the GUIX event queue, and when necessary refreshing or redrawing dirty areas of the GUIX canvas.
8. The serial port driver is initialized by calling the open function:
   A. `err = g_rspi_lcdc.p_api->open(g_rspi_lcdc.p_ctrl, g_rspi_lcdc.p_cfg);`
9. Then it prepares the LCD for receiving image data by calling LCD_init service (defined in lcd_setup.c).

Now the GUIX engine can send the image to the LCD panel.

During this time, user defined callbacks for various components of the widget created in the GUIX Studio project may be invoked. The my_guix_event_handler.c file contains the callback functions for this project, specifically the SplashScreenEventHandler callback. This demonstrates how to use GUIX and GUIX Studio services to control the functionality of the widget at runtime. The callback supplies the event type, which could be a button click or a timer expiration. For the GX_EVENT_SHOW event type (when the splash screen is displayed), this callback starts a timer set to expire in 100 ticks (1 second on a system with a periodic of 100 ticks per second), which will only expire once (is not reset on expiration). Then it returns control to the GUIX system to complete the processing of displaying the splash_screen. When the callback receives the GX_EVENT_TIMER event type from the underlying GUIX engine, it executes a simple change in the splash screen. It overwrites the message from “Version 5.3.3” to “Hello World”.

The image is displayed for a few moments, then the display and serial port driver must be closed for subsequent calls on the guix_driver_test_image_draw service:

1. Stop the display driver:
   `g_display0.p_api->stop(g_display0.p_ctrl);`
2. Close the display driver interface:
   `g_display0.p_api->close(g_display0.p_ctrl);`
3. Close the SPI driver interface:
   `err = g_rspi_lcdc.p_api->close (g_rspi_lcdc.p_ctrl);`

The number of errors are returned to the my_guix_thread_entry function. Zero indicates a successful test.

LCD_TEST

The LCD_TEST is useful as a test of the SPI and display drivers, and LCD hardware; it does not use the GUIX services.

This test requires an LCD SPI callback to be defined. However, it does not use any GUIX services, so the `my_guix_callback` is not used. In LCD_TEST mode, there is a function for a creating simple color bar display, test_LCD defined in `LCD_Setup.c`.
The guix_driver_test_image_draw service in LCD_TEST mode directly opens the display driver and uses the same display driver and serial port interface drivers as it does in GUIX_TEST mode:

1. The serial port driver is initialized by calling `open`:
   `err = g_rspi_lcdc.p_api->open(g_rspi_lcdc.p_ctrl, g_rspi_lcdc.p_cfg);`
2. The display driver is initialized by the `open` call:
   `g_display0.p_api->open(g_display0.p_ctrl, g_display0.p_cfg)'`
3. Then the LCD hardware is prepared for receiving image data by calling the `LCD_init` service.
4. Enable the display by calling:
   `g_display0.p_api->start(g_display0.p_ctrl)`
5. The image drawing function, `test_LCD`, is called and three color bars, red, green, and blue should appear on the LCD screen.

The image is displayed for a few moments, then the display and serial port driver must be closed for subsequent calls on the guix_driver_test_image_draw service:

1. Stop the display driver:
   `g_display0.p_api->stop(g_display0.p_ctrl);`
2. Close the display driver interface:
   `g_display0.p_api->close(g_display0.p_ctrl);`
3. Close the SPI driver interface:
   `err = g_rspi_lcdc.p_api->close (g_rspi_lcdc.p_ctrl);`

The number of errors are returned to the `my_guix_thread_entry` function. Zero indicates a successful test.

After both tests are run, the total number of errors and display events are printed in the debug console if SEMI_HOSTING is defined in the `my_guix_thread_entry` function.

Note: The above description assumes you are familiar with using `printf()` and the debug console in the Synergy Software Package. If you are unfamiliar with this, refer to the “How do I Use Printf() with the Debug Console in the Synergy Software Package” Knowledge Base article, available as described in the Reference section at the end of this document. Alternatively, you can see results via the watch variables in the debug mode.

A few key properties are configured in this application project to support the required operations and the physical properties of the target board and MCU. Below the properties with the values set for this specific project are highlighted in **bold**. You can also open the application project and view these settings in the property window as a hands-on exercise.

Table 12. Configuration for the GUIX Synergy Port Framework Module on GUIX on gx

ISDE Property	Value	Description
Enable Synergy 2D Drawing Engine Support	Yes	If Synergy 2D Drawing Engine (DRW) Support is enabled, the rotation is processed by Synergy DRW with texture mapping. If not enabled, the rotation is processed by software copy.
Enable Synergy JPEG Support	Yes	Enabling this will place restrictions on JPEG image size, width modulus, format, output alignment, and so forth. It also limited to only decoding the JPEG image into the frame buffer. (The software decoder doesn’t have any of those restrictions, but of course it uses more CPU time.)

Table 13. Configuration for the GUIX Synergy Port Framework Module on sf_el_gx

ISDE Property	Value	Description
Parameter Checking	Enabled	Enable the parameter checking.
Name	`g_sf_el_gx0`	Name of SF_EL_GX instance which is generated by ISDE. Name must be a valid C symbol.
Name of Display Driver Run-time Configuration (Must be a valid symbol)	`g_display0_runtime_cfg_bg`	Specify the name of run-time configuration for the DISPLAY module you specified in the Synergy Configuration. Name must be a valid C symbol and NULL is not allowed to be set.
Name of Frame Buffer A (Must be a valid symbol)	`g_display0_fb_background[0]`	Specify the name of frame buffer. A DISPLAY module configuration in the Synergy Configuration contains the name of frame buffer to create. Set the name of frame buffer here. Name must be a valid C symbol and NULL is not allowed to be set.
Name of Frame Buffer B (NULL allowed if consisting a single frame buffer system)	`g_display0_fb_background[1]`	Specify the name of another frame buffer. If you want to design your graphics system with a single frame buffer, set NULL to this parameter, or set same frame buffer name with parameter **Name of Frame Buffer A**. See Tearing in Single Buffer Designs
Name of User Callback function	`my_guix_callback`	Name of User Callback function invoked by the Module when events happen (optional).
Screen Rotation Angle (Clockwise)	0	Angle of screen rotation (degree). If non-zero value is selected, screen rotation is enabled and GUIX draws screen image on a frame buffer, rotating the image with the angle in the counterclockwise way.
GUIX Canvas Buffer (required if rotation angle is not zero)	Not used	If not enabling the screen rotation, set this to Not Used.
Size of JPEG Work Buffer (valid if JPEG hardware acceleration enabled)	81920	The JPEG work buffer size in bytes. Value must be a valid integer value and zero is allowed to be set if JPEG acceleration is not used. Larger buffer size shortens the drawing time. See Size of JPEG Work Buffer
Memory section for GUIX Canvas Buffer	bss	Name of memory section where you want to allocate the GUIX Canvas Buffer. Enter a valid section name defined in the linker script file. Name must be a valid C symbol.
Memory section for JPEG Work Buffer	bss	Name of memory section where you want to allocate the JPEG Work Buffer. Enter a valid section name defined in the linker script file. Name must be a valid C symbol.

Table 14. Configuration for the GLCD HAL Module on r_glcd

ISDE Property	Value	Description
Parameter Checking	Enabled	Enable or disable the parameter checking.
Name	`g_display0`	The name to be used for a GLCDC module control block instance. This name is also used as the prefix of the other variable instances.
Name of display callback function to be defined by user	NULL	Name must be a valid C symbol.
Input - Panel clock source select	Internal clock (GLCDCLK)	Choose the panel clock source depends on your system.
Input - Graphics screen1	Used	Specify "Used" if the graphics screen N is used. Then, the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is autogenerated by ISDE. If not using either of the graphics screens, specify "Not used." Then, the frame buffer is not created. Note that there is no memory read access to the frame buffer when you specify “Not used,” reducing the consumption of bus bandwidth.
Input - Graphics screen1 frame buffer name	fb_background	Custom name for frame buffer.
Input - Number of Graphics screen1 frame buffer	2	Number of graphics selection.
Input - section where Graphics screen1 frame buffer allocated	bss	Specify the section name to allocate the frame buffer. This is valid if "Input - Graphics screen1" is set as "Used."
Input - Graphics screen1 input horizontal size	256	Specify the number of horizontal pixels.
Input - Graphics screen1 vertical size	320	Specify the number of vertical pixels.
Input - Graphics screen1 input horizontal stride (not bytes but pixels)	256	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically, this parameter is set to same number as parameter 'input horizontal size'.
Input - Graphics screen1 input format	16bits ARGB1555, 16bits	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.
Input - Graphics screen1 input line descending	Not used	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off".
Input - Graphics screen1 input line repeat	Off	Specify "On" if expecting to repeatedly read a raster image which is smaller than the LCD panel size. Usually "Off". For details, see the description of Line Repeating function.
Input - Graphics screen1 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.
Input - Graphics screen1 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.
Input - Graphics screen1 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.

ISDE Property	Value	Description
Input - Graphics screen1 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.
Input - Graphics screen1 layer background color Red	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer background color Green	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer background color Blue	255	Specify the background color in the graphics screen N.
Input - Graphics screen1 layer fading control	None	Specify "On" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Off" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input - Graphics screen1 layer fade speed	0	Specify the number of frames for the fading transition to complete.
Input - Graphics screen2	Not used	Specify "Used" if the graphics screen N is used. Then the frame buffer named "display_fb_background" for graphics screen1 and "display_fb_foreground" for graphics screen2 is auto-generated by ISDE. If not using either of the graphics screens, specify "Not used". Then the frame buffer is not created. Note that there is no memory read access to the frame buffer when you specify "Not used", which reduces the consumption of bus bandwidth.
Input - Graphics screen2 frame buffer name	`fb_foreground`	Custom name for frame buffer.
Input - Number of Graphics screen2 frame buffer	2	Number of graphics selection.
Input - section where Graphics screen2 frame buffer allocated	sdram	Specify the section name to allocate the frame buffer. This is valid if "Input - Graphics screen1" is set as "Used."
Input - Graphics screen2 input horizontal size	800	Specify the number of horizontal pixels. Default value is the size for an image with 800x480 pixels
Input - Graphics screen2 vertical size	480	Specify the number of vertical pixels. Default value is the size for an image with 800x480 pixels.

ISDE Property	Value	Description
Input - Graphics screen2 input horizontal stride (not bytes but pixels)	800	Specify the memory stride for a horizontal line. This value must be specified with the number of pixels, not actual bytes. Typically, this parameter is set to same number as parameter 'input horizontal size'. Default value is the size for an image with 800x480 pixels.
Input - Graphics screen2 input format	16bits RGB565	Specify the graphics screen Input format. If selecting CLUT formats, you must write CLUT data using clut before performing start. Default setting supports a RGB565 formatted image.
Input - Graphics screen2 input line descending	Off	Specify "On" if image data descends from the bottom line to the top line in the frame buffer. Usually "Off".
Input - Graphics screen2 input line repeat	Off	Specify "On" if expecting to repeatedly read a raster image which is smaller than the LCD panel size. Usually "Off". For details, see the description of Line Repeating function.
Input - Graphics screen2 input line repeat times	0	Specify the number of repeating times for a raster image which is read repeatedly in a frame.
Input - Graphics screen2 layer coordinate X	0	Specify the horizontal offset in pixels of the graphics screen from the background screen.
Input - Graphics screen2 layer coordinate Y	0	Specify the vertical offset in pixels of the graphics screen from the background screen.
Input - Graphics screen2 layer background color alpha	255	Based on the alpha value, either the graphics screen2 (foreground graphics screen) is blended into the graphics screen1 (background graphics screen) or the graphics screen1 is blended into the monochrome background screen.
Input - Graphics screen2 layer background color Red	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer background color Green	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer background color Blue	255	Specify the background color in the graphics screen N.
Input - Graphics screen2 layer fading control	None	Specify "On" when performing a fade-in for the graphics screen. The transparent screen changes gradually to opaque. Specify "Off" when performing the fade-out for the graphics screen. The opaque screen changes gradually to transparent. Note that this processing is accelerated by the GLCDC hardware and cannot stop once started. The transition status can be monitored by statusGet.
Input - Graphics screen2 layer fade speed	0	Specify the number of frames for the fading transition to complete.
Output - Horizontal total cycles	320	Specify the total cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.

ISDE Property	Value	Description
Output - Horizontal active video cycles	240	Specify the number of active video cycles in a horizontal line. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal back porch cycles	6	Specify the number of back porch cycles in a horizontal line. Back porch starts from the beginning of Hsync cycles, which means back porch cycles contain Hsync cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal sync signal cycles	4	Specify the number of Hsync signal assertion cycles. Set to the number of cycles defined in the data sheet of LCD panel sheet in your system. Default value matches LCD panel on S7G2 PE-HMI1 board.
Output - Horizontal sync signal polarity	Low active	Select the polarity of Hsync signal to match your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical total lines	328	Specify number of total lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical active video lines	320	Specify the number of active video lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical back porch lines	4	Specify the number of back porch lines in a frame. Back porch starts from the beginning of Vsync lines, which means back porch lines contain Vsync lines. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical sync signal lines	4	Specify the Vsync signal assertion lines in a frame. Set to the number of lines defined in the data sheet of LCD panel sheet in your system. Default value matches the LCD panel on S7G2 PE-HMI1 board.
Output - Vertical sync signal polarity	Low active	Select the polarity of Vsync signal to match to your system. Default setting matches LCD panel on S7G2 PE-HMI1 board.
Output - Format	16bits RGB565	Specify the graphics screen output format to match to your LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Endian	Little endian	Select data endian for output signal to LCD panel. Default setting matches the LCD panel on S7G2 PE-HMI1 board.

ISDE Property	Value	Description
Output - Color order	RGB	Select data order for output signal to LCD panel. The order of blue and red can be swapped if needed. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Data Enable Signal Polarity	High active	Select the polarity of Data Enable signal to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Sync edge	Rising Edge	Select the polarity of Sync signals to match to your system. Default setting matches the LCD panel on S7G2 PE-HMI1 board.
Output - Background color alpha channel	255	Specify the background color of the background screens.
Output - Background color R channel	0	Specify the background color of the background screens.
Output - Background color G channel	0	Specify the background color of the background screens.
Output - Background color B channel	0	Specify the background color of the background screens.
CLUT	Not used	Specify "Used" if selecting CLUT formats for a graphics screen input format. Then, a buffer named "CLUT_buffer" for the CLUT source data is generated in the ISDE auto-generated source file.
CLUT - CLUT buffer size	256	Specify the number of entries for the CLUT source data buffer. Each entries consume 4 bytes (1 word). Words of CLUT source data specified by this parameter are generated in the ISDE auto-generated source file.
TCON - Hsync pin select	LCD_TCON0	Select the TCON pin used for the Hsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - Vsync pin select	LCD_TCON1	Select TCON pin used for Vsync signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - DataEnable pin select	LCD_TCON2	Select TCON pin used for DataEnable signal to match to your system. Default setting is for LCD panel on S7G2 PE-HMI1 board.
TCON - Panel clock division ratio	1/32	Select the clock source divider value. See the note at bottom of this table about the source clock for the pixel clock.
Color correction - Brightness	Off	Specify "On" when performing brightness control. If specifying "Off", the setting below does not affect the output color.
Color correction - Brightness R channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Brightness G channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Brightness B channel	512	Output color level is calculated as follows: Output color level = Input color level +/ - 512. Set the value for each of R, G, B channels.
Color correction - Contrast	Off	Specify "On" when performing contrast control. If specifying "Off", the setting below does not affect the output color.

ISDE Property	Value	Description
Color correction - Contrast(gain) R channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Contrast(gain) G channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Contrast(gain) B channel	128	Output color level is calculated as follows: Output color level = Input color level x (/128). Set the value for each of R, G, B channels.
Color correction - Gamma correction(Red)	Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the red channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain R[0-15]	0	Set the gain value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold R[0-15]	0	Set the threshold value for the red channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level/1024 (/128).
Color correction - Gamma correction(Green)	Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the green channel. If specifying "Off", the settings for gain and threshold do not affect the output color.
Color correction - Gamma gain G[0-15]	0	Set the gain value for the green channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level / 1024 (/128).
Color correction - Gamma threshold G[0-15]	0	Set the threshold value for the green channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level/1024 (/128).
Color correction - Gamma correction(Blue)	Off	Control for each channel R/G/B. Specify "On" when performing gamma correction for the blue channel. If specifying "Off", the settings for gain and threshold do not affect the output color.

ISDE Property	Value	Description
Color correction - Gamma gain B[0-15]	0	Set the gain value for the blue channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between ((Gamma threshold R[N-1])<<2) and ((Gamma threshold R[N])<<2). The output value is calculated as follows: Output color level = Input color level/1024 (/128).
Color correction - Gamma threshold B[0-15]	0	Set the threshold value for the blue channel in the area N on the gamma correction curve. The gain setting for area N is applied to the input data with a color level between Gamma threshold R[N-1] and Gamma threshold R[N]. The output value is calculated as follows: Output color level = Input color level/1024 (/128).
Dithering	Off	Dithering enable. Specify "On" when applying the dither effect to reduce the banding in case of selecting RGB666 or RGB565 output formats. Dithering can be applied when converting. If specified "Off", the settings for dithering below do not affect the output. For details on the dither effect, see Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Mode	Truncate	Specify the dither mode. For detail, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern A	Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHA) in the hardware manual.
Dithering - Pattern B	Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHB) in the hardware manual.
Dithering - Pattern C	Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHC) in the hardware manual.
Dithering - Pattern D	Pattern 11	Specify the dither pattern for 2X2 pattern mode. For details, see the Output Control Block Panel Dither Correction Register (OUT_PDTHD) in the hardware manual.
Misc - Correction Process Order	Brightness and Contrast then Gamma	Specify the color correction processing order if needed.
Line Detect Interrupt Priority	Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX)	The driver needs valid interrupt priority setting and it won't work if disabled.
Underflow 1 Interrupt Priority	Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX)	The driver needs valid interrupt priority setting and it won't work if disabled.
Underflow 2 Interrupt Priority	Disabled	The driver needs valid interrupt priority setting and it won't work if disabled.

Table 15. Configuration for the JPEG Decode Framework Module on sf_jpeg_decode

ISDE Property	Value	Description
Parameter Checking	Enabled	Enable or disable the parameter checking.
Name	g_sf_jpeg_decode0	The name to be used for a JPEG Decode Framework module instance.

Table 16. Configuration for the JPEG Decode HAL Module on r_jpeg

ISDE Property	Value	Description
Parameter Checking	Enabled	Enable or disable the parameter error checking.
Name	g_jpeg_decode0	The name to be used for a JPEG Decode module instance.
Byte Order for Input Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8)	Specify the byte order for input data. The order is swapped as specified in every 8-byte.
Byte Order for Output Data Format	Normal byte order (1)(2)(3)(4)(5)(6)(7)(8)	Specify the byte order for output data. The order is swapped as specified in every 8-byte.
Output Data Color Format	Pixel Data RGB565 format, Pixel Data ARGBB888 format Default: Pixel Data RGB565 format	Specify the output data format.
Alpha value to be applied to decoded pixel data (only valid for ARGB8888 format)	255	Specify the alpha value for the output data format (only valid for ARGB8888 format).
Name of user callback function	NULL	Specify the name of user callback function.
Decompression Interrupt Priority	Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX)	Decompression interrupt priority selection.
Data Transfer Interrupt Priority	Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX)	Data transfer interrupt priority selection.

Table 17. Configuration for the D/AVE 2D Driver on dave2d

ISDE Property	Value	Description
No configurable settings		

Table 18. Configuration for the D/AVE 2D Port on sf_tes_2d_drw

ISDE Property	Value	Description
Work memory size for display lists in bytes	32768	Work memory size for display lists selection
DRW Interrupt Priority	Priority 3 (CM4: valid, CM0+: lowest- not valid if using ThreadX)	DRW INT selection

Table 19. Configuration for the SPI driver

ISDE Property	Value Set
Clock Phase	Data sampling on even edge, data variation on odd edge
Clock Polarity	High when idle
Callback	my_lcd_spi_callback
SPI Mode	Clock Synchronous operation
Receive Interrupt Priority	Priority 2
Transmit Interrupt Priority	Priority 2
Transmit End Interrupt Priority	Priority 12
Error Interrupt Priority	Priority 2

The RSPI bus pins need to be configured. In the Thread Stack pane, select the **Pins** tab > **Peripherals – Connectivity:SPI** > **SPI0**. Change the **Operation Mode** to **Enabled**. Verify the pin assignments as shown in Figure 10:

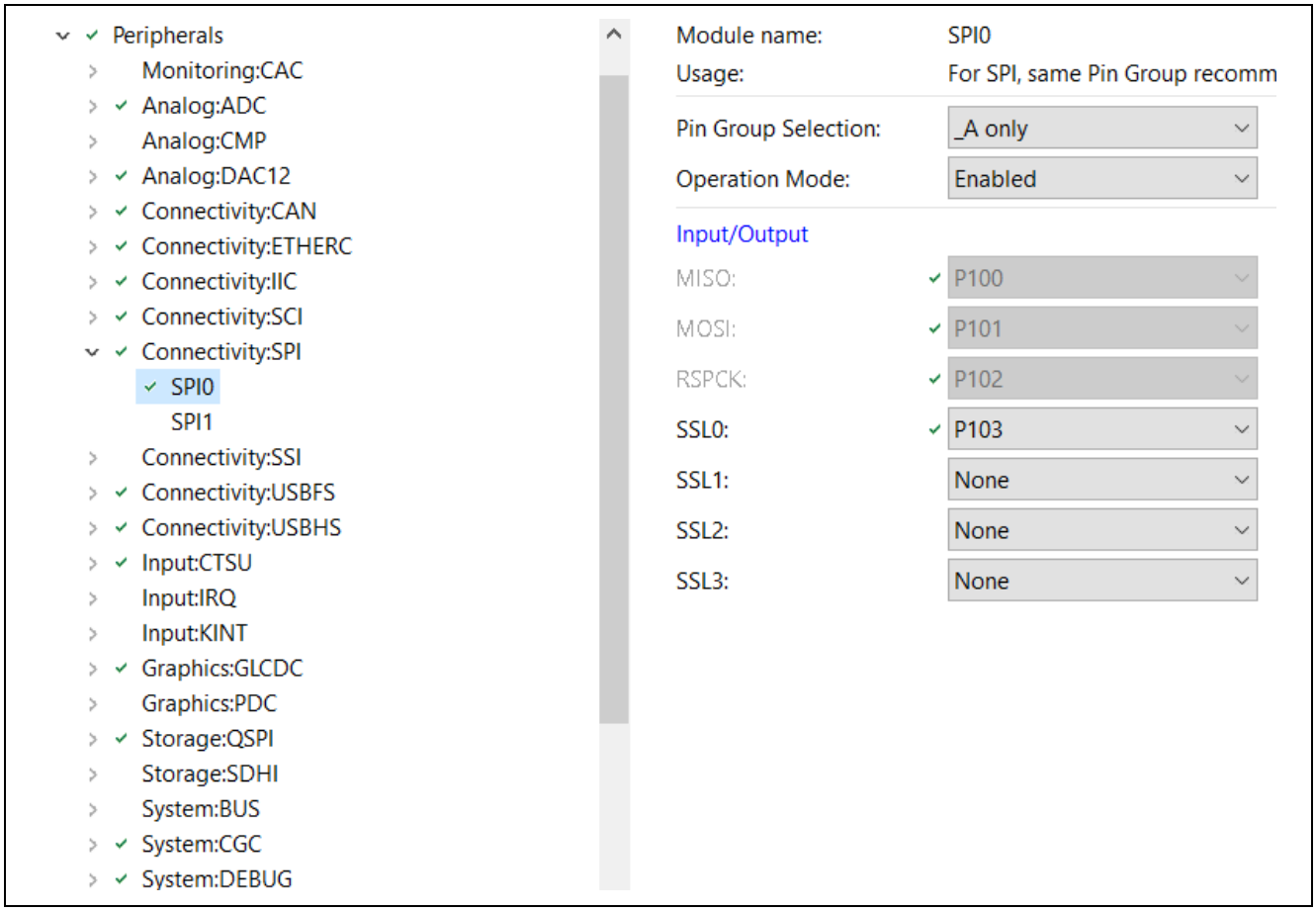

Figure 10. Configuration of RSPI bus pins

Several LCD panel signals are controlled by GPIO pins under application control. Because the RSPI is configured for Clock Synchronous operation, the SPI slave select signal must be controlled by the application as a GPIO pin. Slave select for the LCD is on P611. Additionally, there is a reset pin on P610 and a command pin on P115. We need to enable these three pins as GPIO outputs.

Select **Pins -> P6 -> P610**. Set the **Mode** to **Output mode (Initial High)** as Figure 11 shows. Repeat for P610 and P115.

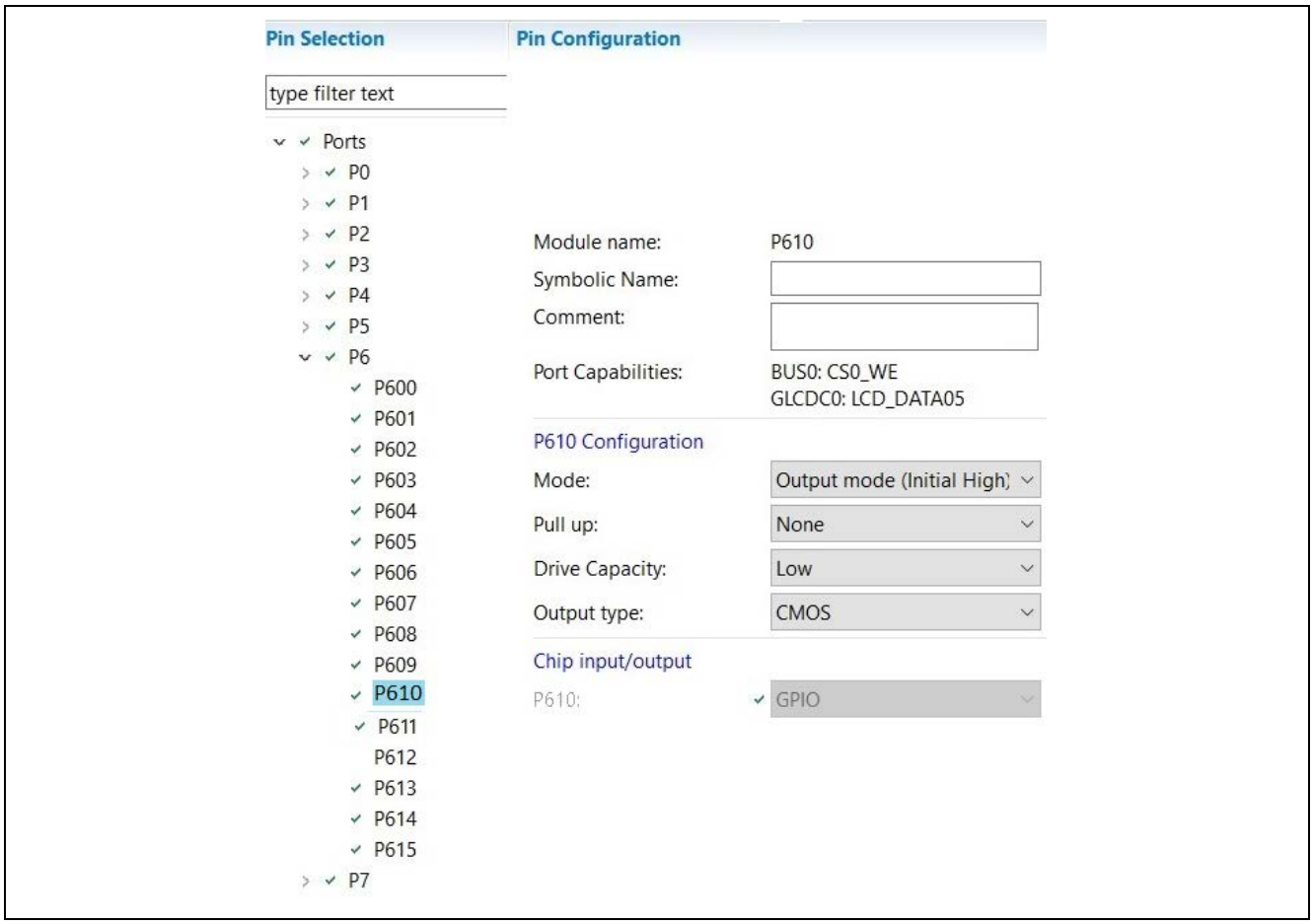

Figure 11. Configuration of GPIO pins for the LCD Panel

8. Customizing the GUIX Synergy Port Framework Module for a Target Application

The project can be customized with the existing splash screen and memory resources. For example, the screen rotation angle can be modified to orient the image portrait or landscape. The project can use the hardware JPEG decoder, for optimal speed, or if memory is an issue to use the software decoder. See section 3.1.1 for more details on how to do this.

An application can be created without using GUIX Studio. To do so, the application simply calls the `gx_<widget_type>_create()` function directly, and manually creates all the child widgets. That is a lot of work, so for anything other than a highly specialized or trivial application, most developers prefer to use GUIX Studio. The LCD_TEST in the project is also an example of manually drawing an image.

To understand the GUIX system, and step through the GUIX source code logic, click to add **GUIX Source** under GUIX stacks, this adds the source code into project as shown in Figure 12. The warning on the GUIX Source can be ignored or disabled by setting **Show linkage warning** in the properties. Next, generate the project, rebuild it, and debug the application.

For best results stepping through code set the optimization level to -O0 (no optimization). Right-click the project, choose **Properties -> C/C++ Build (expand) -> Settings -> Optimization**.

Note: The DTC module is optional. If it gets added, remove it.

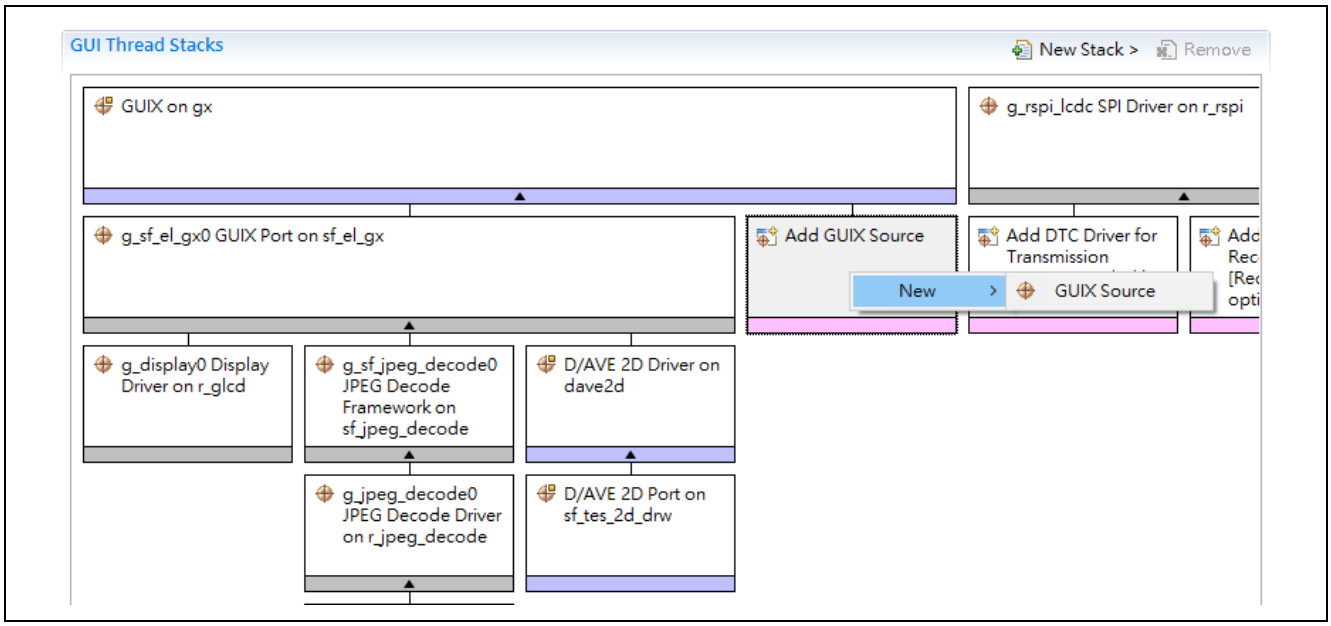

Figure 12. Add GUIX Source code

9. Running the GUIX™ Synergy Port Framework Module Application Project

To run the GUIX Synergy Port Framework module application project and to see it executed on a target kit, you can simply import it into your ISDE, compile, and run debug. Refer to the *Renesas Synergy™ Project Import Guide* (r11an0023eu0121-synergy-ssp-import-guide.pdf, included in this package) for instructions on importing the project into e^2 studio or IAR embedded workbench and building/running the application.

To implement the GUIX Synergy Port Framework module application in a new project, follow the steps below for defining, configuring, auto-generating files, adding code, compiling, and debugging on the target kit. Following these steps is a hands-on approach that can help make the development process with SSP more practical.

Note: The following steps are described in sufficient detail for someone experienced with the basic flow through the Synergy development process. If these steps are not familiar, refer to the first few chapters of the *SSP User's Manual* for a description of how to accomplish these steps.

To create and run the GUIX Synergy Port Framework application project, simply follow these steps:

1. Create a new Renesas Synergy project for the SK-S7G2 called `GUIX_sf_el_gx_AP`. Choose BSP only.
2. Select the **Threads** tab.
3. Check if the CGC is added to the HAL/Common thread stack. If not, add it.
4. Add a thread called my_guix_thread and set the priority to 10.
5. Add a semaphore under GUI Thread: g_my_gui_semaphore.
6. Add an instance of the GUIX driver: Click on the **(+)** icon in the **Thread Stack** pane and choose **X-ware -> GUIX -> GUIX** on gx. This will add the GUIX driver `sf_el_gx`, the display driver `r_glcd`, the D/AVE 2D driver, and the JPEG decoder framework. Keep the default names of each instance.
7. Add the serial interface to the LCD screen by choosing **(+) -> Driver -> Connectivity -> SPI Driver** on r_rspi. Rename it to `g_rspi_lcdc`.
8. Click on the **Generate Project Content** button.
9. Add the code from the supplied project file `guix_driver_sf_el_gx_mg_ap.c` and `guix_driver_sf_el_gx_mg_ap.c` or copy the logic from these files to the generated thread entry file. If you named your thread `my_guix_thread`, then the Synergy generated thread entry file will be

`my_guix_thread_entry.c`. Add the four files generated by GUIX Studio as described previously in section 7 under GUIX TEST.

10. The project has an optional event handler for the main window, `SplashScreenHandler`. If GUIX studio specification files specify such a callback, the application must define it. Use the event handler function, `SplashScreenEventHandler`, in the `my_guix_event_handler.c` project file for this callback (or define your own).
11. Set the Thread Stack component (for example, `g_sf_el_gx0` driver instance and `r_lcd` display driver) properties as described in tables in section 7.
12. Set the pin and peripheral settings as described in section 7.
13. Call the `guix_driver_test_image_draw` service with the desired mode, LCD_TEST or GUIX_TEST to process an image to the LCD panel from the thread entry function `my_guix_thread_entry` (if your thread is named `my_guix_thread`).
14. For printf debug output, one can either #define SEMI_HOSTING in the thread entry function or for the whole project. For the latter, right click on the project **-> properties -> C/C++ Build (expand the list) -> Cross ARM C Compiler -> Preprocessor**. Click the **(+)** and enter SEMI_HOSTING. For IAR, right-click on the project **Options -> C/C++ compiler -> Preprocessor** and add SEMI_HOSTING.
15. Start to debug the application.
16. The output can be viewed in the Renesas Debug Console. If no error occurred, the output will be as shown in the following screenshot.

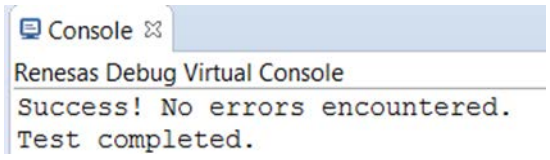

Figure 13. Example Output from GUIX Synergy Port Framework Application Project

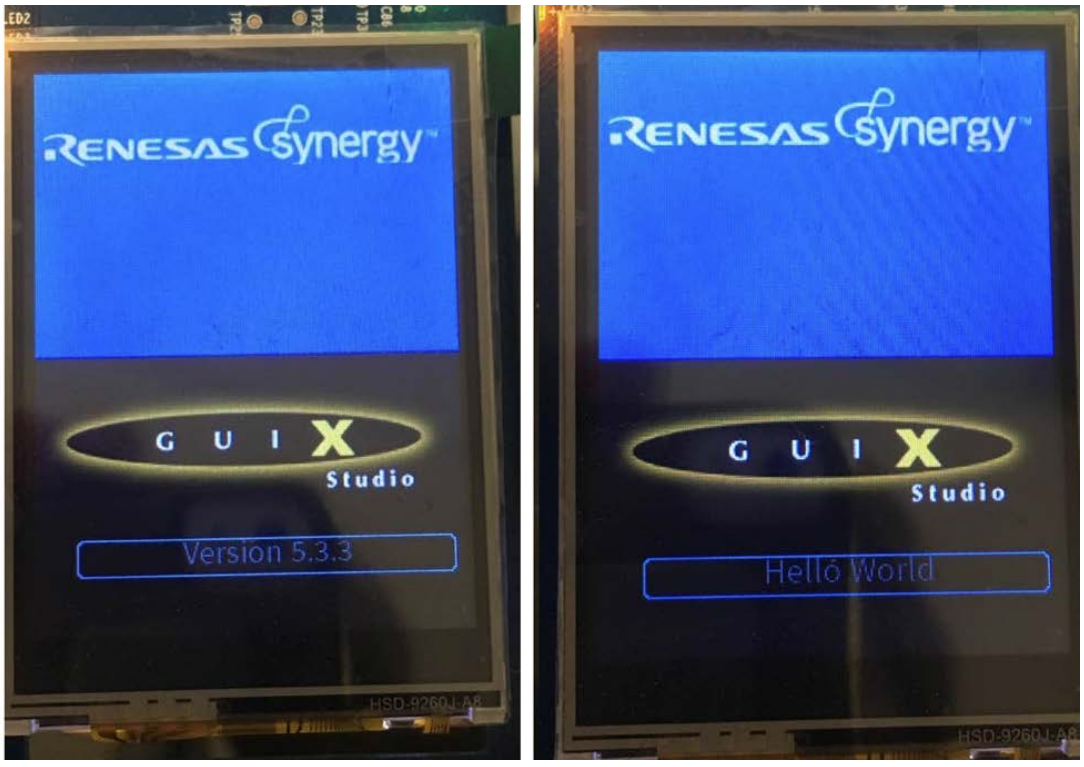

Figure 14. LCD display from GUIX SF_EL_GX Driver Application Project in GUIX_TEST Mode

The screen on the left appears initially for about 1 second. Then the text at the bottom of the screen changes to “Hello World” on the right.

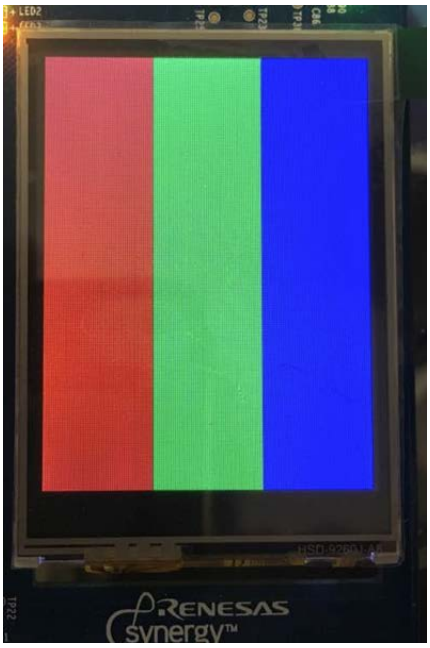

Figure 15. LCD display from GUIX SF_EL_GX Driver Application Project in LCD_TEST Mode

10. GUIX™ Synergy Port Framework Module Conclusion

This module guide has provided all the background information needed to select, add, configure, and use the module in an example project. The project walks you through the process of initializing the GUIX system and hardware drivers and setting the drawing component properties and pin assignments in the e^2 studio project pane. The project demonstrates how to draw an image to the LCD screen directly in LCD_TEST mode which is also a simple test of the GUIX system and the hardware. The project also demonstrates how to create resource and specification files from GUIX Studio for more complex LCD screens.

Many of these steps were time consuming and error-prone activities in previous generations of embedded systems. The Renesas Synergy™ Platform makes these steps much less time consuming and removes the common errors, like conflicting configuration settings or the incorrect selection of low-level modules. The use of high-level APIs (as demonstrated in the application project) illustrates additional development-time savings by allowing work to begin at a high level and avoiding the time required in older development environments to use, or, in some cases, create, lower-level drivers.

11. GUIX™ Synergy Port Framework Module Next Steps

After you have mastered a simple GUIX module project, you may want to review a more complex example.

The resource and specification files generated in GUIX Studio are for fairly simple screens. Buttons, drop-down lists, child windows, and other screen objects may be added. More complex handlers and timers will also greater functionality for an application.

The size and location of memory for the GUIX canvas and decoding JPEG images can also be optimized for greater performance.

You can also add a touchscreen component to the project and write handlers for touch screen events.

12. GUIX™ Synergy Port Framework Module Reference Information

SSP User Manual: Available in html format in the SSP distribution package and as a pdf from the Synergy Gallery.

Links to all the most up-to-date `sf_el_gx` module reference materials and resources are available on the Synergy Knowledge Base: www.renesas.com/knowledgeBase/16977541

Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	www.renesas.com/synergy/software
Synergy Software Package	www.renesas.com/synergy/ssp
Software add-ons	www.renesas.com/synergy/addons
Software glossary	www.renesas.com/synergy/softwareglossary
Development tools	www.renesas.com/synergy/tools
Synergy Hardware	www.renesas.com/synergy/hardware
Microcontrollers	www.renesas.com/synergy/mcus
MCU glossary	www.renesas.com/synergy/mcuglossary
Parametric search	www.renesas.com/synergy/parametric
Kits	www.renesas.com/synergy/kits
Synergy Solutions Gallery	www.renesas.com/synergy/solutionsgallery
Partner projects	www.renesas.com/synergy/partnerprojects
Application projects	www.renesas.com/synergy/applicationprojects
Self-service support resources:	
Documentation	www.renesas.com/synergy/docs
Knowledgebase	www.renesas.com/synergy/knowledgebase
Forums	www.renesas.com/synergy/forum
Training	www.renesas.com/synergy/training
Videos	www.renesas.com/synergy/videos
Chat and web ticket	www.renesas.com/synergy/resourcelibrary

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	May.15.17	-	Initial Release
1.01	Dec.01.17	-	Update to Hardware and Software Resources Table
1.02	May.06.19	-	Updated for SSP v1.6.0. Added note for DTC module.
1.03	Oct.11.19	-	Updated for SSP v1.7.0

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.